VU VRIJE UNIVERSITEIT AMSTERDAM

VU Research Portal

A graph-based formulation for the shift rostering problem

Lai, David; Leung, Janny M.Y.; Dullaert, Wout; Marques, Inês

published in
European Journal of Operational Research
2020

DOI (link to publisher)
10.1016/j.ejor.2019.12.019

document version
Publisher's PDF, also known as Version of record

document license
Article 25fa Dutch Copyright Act

Link to publication in VU Research Portal

citation for published version (APA)
Lai, D., Leung, J. M. Y., Dullaert, W., & Marques, I. (2020). A graph-based formulation for the shift rostering problem. *European Journal of Operational Research*, *284*(1), 285-300. https://doi.org/10.1016/j.ejor.2019.12.019

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

E-mail address:
vuresearchportal.ub@vu.nl

Download date: 05. May. 2023

European Journal of Operational Research 284 (2020) 285–300

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Decision Support

A graph-based formulation for the shift rostering problem

David S. W. Lai[a,*], Janny M. Y. Leung[b], Wout Dullaert[a], Inês Marques[c]

[a] Department of Supply Chain Analytics, Vrije Universiteit Amsterdam, Netherlands
[b] School of Science and Engineering, The Chinese University of Hong Kong, Shenzhen, China
[c] Centre for Management Studies, Instituto Superior Técnico, University of Lisbon, Av. Rovisco Pais 1, Lisbon 1049-001, Portugal

ARTICLE INFO

Article history:
Received 9 March 2018
Accepted 11 December 2019
Available online 16 December 2019

Keywords:
Human resource planning
Shift rostering
Personnel rostering
Scheduling
Network model

ABSTRACT

This paper investigates a shift rostering problem – the assignment of staff to shifts over a planning horizon such that work rules are observed. Traditional integer-programming models are not able to solve shift rostering problems effectively for large number of staff and feasible shift patterns. We formulate work rules in terms of newly-proposed prohibited meta-sequences and resource constraints. A graph-based formulation and a specialized graph construction algorithm are proposed where the set of feasible shift patterns is represented by paths of a graph. The formulation size depends on the structure of the work-rule constraints and is independent of the number of staff. This approach results in smaller networks allowing large-scale rostering problems with hard constraints to be solved efficiently using standard commercial solvers. Moreover, it allows finding multiple optimal solutions which are beneficial for managerial decision makers. Computational results show that the proposed approach can obtain new best-known solutions and identify proven optimal solutions for almost all NSPLIB instances at significantly lower CPU times.

© 2019 Elsevier B.V. All rights reserved.

1. Introduction

Personnel scheduling or rostering problems are an active field of operations research. Complex personnel rostering problems occur in a wide variety of industries and contexts that challenge existing models and solution frameworks. As a result, the academic literature introduces ever-richer personnel rostering problems featuring new side constraints and/or objective functions. In turn, these new problems have triggered the development of more sophisticated solution methods, both exact and heuristic ones (see e.g. Bruecker, Beliën, Van den Bergh, & Demeulemeester, 2018; Doi, Nishi, & Voss, 2018; Dolgui, Kovalev, Kovalyov, Malyutin, & Soukhal, 2018; Rahimian, Akartunali, & Levine, 2017) to better support real-life applications in different contexts such as health care delivery (e.g. Fügener, Pahr, & Brunner, 2018; Vermuyten, Rosa, Marques, Beliën, & Barbosa-Póvoa, 2018), project management (e.g. Maenhout and Vanhoucke (2016); Van Den Eeckhout, Maenhout, and Vanhoucke (2019)), airline sector (e.g. Bruecker et al. (2018); Doi et al. (2018)), call-centers (e.g. Örmeci, Salman, and Yücel (2014); Taskiran and Zhang (2017)) and production systems (e.g. Dolgui et al. (2018)). For a discussion of classification methods for personnel scheduling problems the reader is referred to the most recent literature review on this area of research (Van den Bergh, Beliën, Bruecker, Demeulemeester, & Boeck, 2013).

Various literature reviews have tried to classify the growing body of literature (see e.g. Blöchliger, 2004; Ernst, Jiang, Krishnamoorthy, & Sier, 2004; Van den Bergh et al., 2013) and offer specific guidelines on how to advance research on rostering problems. Ernst et al. (2004) show that a large amount of work has been done in rostering and personnel scheduling, but also highlight significant room for improvement. One of the main research avenues demanding further work was the generalization of models and methods. The authors observe that models and algorithms often need significant modifications when they are to be transferred to a different application area or to accommodate changes within an organization. To generalize research efforts on rostering problems, the authors suggest the formulation of new models that provide more flexibility to accommodate individual workplace practices. This can then lead to the development of more general algorithms that are more robust to changes in the rostering requirements.

Similar concerns are expressed by Van den Bergh et al. (2013) almost a decade later. The authors show that personnel scheduling problems come in many variations involving several hard and soft constraints. Yet, the effect of those constraints on model complexity has remained barely studied. A more dedicated theoretic study is suggested to understand the effect of the different constraints and to give the possibility to develop well-suited

* Corresponding author.
E-mail addresses: david.lai@vu.nl (D.S.W. Lai), jannyleung@cuhk.edu.cn (J.M.Y. Leung), wout.dullaert@vu.nl (W. Dullaert), ines.marques.p@tecnico.ulisboa.pt (I. Marques).

https://doi.org/10.1016/j.ejor.2019.12.019
0377-2217/© 2019 Elsevier B.V. All rights reserved.

algorithms. Similarly, Smet, Brucker, Causmaecker, and Vanden Berghe (2016a) note that academic advances in rostering and personnel scheduling mainly focus on solving specific variants of this problem using intricate exact or (meta)heuristic algorithms, while little attention has been devoted to studying the underlying structure of the problems.

Answering the appeal by Ernst et al. (2004) to underpin the theory of personnel scheduling, Brucker, Qu, and Burke (2011) are the first authors to systematically study personnel scheduling from a theoretical point of view. The authors propose a general mathematical model which accommodates various characteristics of a rostering problem and identify two polynomially solvable cases. The first considers different shifts which require a constant number of employees on different days. The problem can be solved as a series of transshipment problems. The second case assumes only a single shift type in which the demand of each task is constant, and each employee is available in all periods. This case is reformulated as a minimum cost network flow problem. Brucker et al. (2011) reemphasize the need for more theoretical studies on models and complexity as a promising research line.

Following the call of Brucker et al. (2011), Smet et al. (2016a) extend the research on identifying rostering problems that can be solved in polynomial time by presenting minimum cost network flow formulations for several additional personnel rostering cases. These models consider counter and succession constraints (e.g. the number of days worked, the number of shifts worked of each type). For future research, Smet et al. (2016a) recommend researchers to turn attention towards problems with generalized constraints, e.g. restrictions on consecutive assignments, weekends. The authors also state that models with such intricate constraints, can possibly no longer be transformed into the minimum cost flow problems presented in their paper.

Brucker et al. (2011) already noted that personnel scheduling problems can be formulated as integer linear programs. For some small-sized instances, LP-solvers have been used to solve rostering problems (e.g. Bard, Binici, & deSilva, 2003; Cuevas, Ferrer, Klapp, & Muñoz, 2016; Örmeci et al., 2014). Complex problems can be solved by using heuristics which combine local search and network flow techniques. Over the last decades different mathematical programming approaches and metaheuristics have been developed to solve rostering and personnel scheduling problems. Mathematical programming approaches include branch-and-bound (e.g. Maenhout & Vanhoucke, 2016) and branch-and-price (e.g. Bard & Purnomo, 2005). Recent metaheuristic approaches include iterated local search (e.g. Van Den Eeckhout et al., 2019) and variable neighborhood search (e.g. Smet, Ernst, & Berghe, 2016b; Vermuyten et al., 2018; Zheng, Liu, & Gong, 2017). Decomposition algorithms (e.g. Bruecker et al., 2018; Doi et al., 2018; Taskiran & Zhang, 2017; Wong, Xu, & Chin, 2014) and hybrid techniques (e.g. Andersen, Nielsen, Reinhardt, & Stidsen, 2019; Dahmen & Rekik, 2015; Rahimian et al., 2017; Santos, Toffolo, Gomes, & Ribas, 2016) receive more and more attention from researchers trying to deal with heavily constrained personnel scheduling problems. Graph-based formulations are also used in these decomposition and hybrid approaches. Jarray (2009) solve a maximum flow problem in an exact decomposition approach for an employee days-off scheduling problem of a homogeneous workforce. Côté, Gendron, and Rousseau (2007) introduce a new modeling approach that integrates constraint programming with a graph structure to handle succession constraints. The computational experiments were performed on an employee timetabling problem significantly decreasing computational times of a traditional mixed-integer programming model. Additionally, Cappanera and Gallo (2004) formulate an airline crew rostering problem as a multicommodity flow problem where each employee corresponds to a commodity. In their network, a monthly schedule for an employee is obtained by computing a path in a graph. The dimension of the graph is reduced by a preprocessing phase. Valid inequalities are proposed to tighten the linear programming formulation of the model of our model and to increase computational effectiveness.

The current paper answers the call from Smet et al. (2016a) by studying shift rostering problems (SRPs) with generalized work-rule constraints (such as the consecutive assignments) while also handling resource constraints including counter and succession constraints. Coverage requirements are modeled as soft constraints while work rules are handled as hard constraints. A novel graph-based approach is proposed which allows large-scale rostering problems with hard constraints to be solved efficiently using standard solvers. Traditional integer programming formulations for the staff rostering problem cannot be solved efficiently if there is a large number of staff and/or a large number of feasible shift patterns involved. In this paper, a novel graph-based approach is proposed which allows large-scale rostering problems with hard constraints to be solved efficiently using standard solvers. Work rules are formulated in terms of prohibited meta-sequences and resource constraints offering flexibility for modelling complicated work rules found in practice. In the proposed graph-based formulation, the set of feasible shift patterns is represented by paths of a graph. The graph is constructed by an efficient algorithm for handling the prohibited meta-sequences. As the size of resulting graph depends on the structure of the work rules and is independent on the number of staff, the novel graph-based formulation enables large instances to be solved by standard commercial solvers. Computational experiments on problem instances from the literature demonstrate how the proposed model formulation results in small networks and allows (multiple) optimal solutions to be obtained significantly faster than traditional formulations for problems of real-life scale.

The remainder of this paper is structured as follows. Section 2 introduces the shift rostering problem and presents the traditional 3-index integer linear formulation. A new graph-based formulation is proposed in Section 3, and a new algorithm to generate the graph is described in Section 4. In Section 5, the graph-based approach is tested on randomly generated instances with work rules drawn from practice and problem instances from the literature. Finally, Section 6 concludes the paper.

2. The shift rostering problem

Let $\mathcal{I} = \{1, 2, ..., I\}$ be the set of shifts, $\mathcal{J} = \{1, 2, ..., J\}$ the set of days, and $\mathcal{K} = \{1, 2, ..., K\}$ the set of available staff. For each shift $i \in \mathcal{I}$ and day $j \in \mathcal{J}$, the demand (number of staff required) is defined by $d_{ij} \in \mathbb{Z}^+$. Auxiliary variables o_{ij} and u_{ij} represent the number of excessive staff assigned and extra staff required to cover the demand (overstaff and understaff), respectively. Under- and over-staffing comes at a penalty cost α_{ij}, $\beta_{ij} \in \mathbb{R}^+$, respectively. A 3-index decision variable x_{ij}^k is defined as

$$x_{ij}^k = \begin{cases} 1, & \text{if staff member } k \in \mathcal{K} \text{ is assigned to cover shift} \\ & \quad i \in \mathcal{I} \text{ on day } j \in \mathcal{J}; \\ 0, & \text{otherwise.} \end{cases}$$

and a traditional 3-index model ($SRP1$) can be formulated in (1)–(6) (see e.g. Brucker et al., 2011).

$$(SRP1): \quad min \sum_{i\in\mathcal{I}} \sum_{j\in\mathcal{J}} (\alpha_{ij}\, u_{ij} + \beta_{ij}\, o_{ij}) \tag{1}$$

$$s.t. \sum_{k\in\mathcal{K}} x_{ij}^k + u_{ij} - o_{ij} = d_{ij}, \qquad \forall i \in \mathcal{I}, j \in \mathcal{J}, \tag{2}$$

$$u_{ij}, o_{ij} \geq 0, \qquad \forall i \in \mathcal{I}, j \in \mathcal{J}, \tag{3}$$

$$x_{ij}^k \in \{0, 1\}, \qquad \forall i \in \mathcal{I}, j \in \mathcal{J}, k \in \mathcal{K}, \tag{4}$$

$$\sum_{i\in\mathcal{I}} x_{ij}^k = 1, \qquad \forall j \in \mathcal{J}, k \in \mathcal{K}, \tag{5}$$

Work-rule constraints. (6)

The objective function (1) minimizes the sum of under- and over-staffing penalties. Constraints (2) are the coverage requirements. As the demand d_{ij} is integer, both under- and over-staffing (u_{ij} and o_{ij}) are guaranteed to be integer, and only non-negativity restrictions (3) are needed. Constraints (4) model the binary domain for variables x_{ij}^k. Exactly one shift in a day should be assigned to each staff member (5). Other work rules are modeled as hard constraints (6).

In the literature there is a wide variety of hard constraints that commonly appear in rostering problems under consideration. To illustrate how they are traditionally formulated consider the case where a working shift can be a morning shift, an afternoon shift or a night shift. Let M be the set of morning shifts, A be the set of afternoon shifts, N be the set of night shifts, $W = M \cup A \cup N$ be the working shifts, and D be the set containing a shift which represents day-off.

(a) *a day-off should be assigned after a night shift.*

$$\sum_{i\in N} x_{i,j}^k + \sum_{i\in W} x_{i,j+1}^k \le 1, \qquad \forall j \in \mathcal{J}, k \in \mathcal{K} : j \le |\mathcal{J}| - 1. \tag{6a}$$

(b) *no night-dayoff-night shift sequence.*

$$\sum_{i\in N} x_{i,j}^k + \sum_{i\in D} x_{i,j+1}^k + \sum_{i\in N} x_{i,j+2}^k \le 2, \quad \forall j \in \mathcal{J}, k \in \mathcal{K} : j \le |\mathcal{J}| - 2. \tag{6b}$$

(c) *at most 2 consecutive working shifts.*

$$\sum_{i\in W} x_{i,j}^k + \sum_{i\in W} x_{i,j+1}^k + \sum_{i\in W} x_{i,j+2}^k \le 2, \quad \forall j \in \mathcal{J}, k \in \mathcal{K} : j \le |\mathcal{J}| - 2. \tag{6c}$$

(d) *at least 2 consecutive working shifts.*

$$\sum_{i\in D} x_{i,j}^k + \sum_{i\in W} x_{i,j+1}^k + \sum_{i\in D} x_{i,j+2}^k \le 2, \quad \forall j \in \mathcal{J}, k \in \mathcal{K} : j \le |\mathcal{J}| - 2. \tag{6d}$$

The planning horizon for shift assignment and rostering is often quite long. Horizons between 2 and 52 weeks are usually considered with several shifts in a day (see e.g. Burke & Curtois, 2014; Vanhoucke & Maenhout, 2007; Vanhoucke & Maenhout, 2009). As the number of variables and constraints of *SRP*1 also increase with the number of staff, the resulting shift rostering problems are large-scale mixed-integer programs which are difficult to solve due to the work rule constraints, as observed by e.g. Brunner, Bard, and Köhler (2013) and Lau (1996) for related staff rostering problems. When the staff is homogeneous, the resulting symmetries in assignment often further increase the solution time in a branch-and-bound framework (Margot, 2010). While recognizing that heterogeneity in staff skills is an important consideration in many shift and tour scheduling problems, we focus on the setting where staff can be considered identical (as is also the case in Jarray, 2009; Maenhout & Vanhoucke, 2016). In the next section, we propose a graph-based formulation that turns homogeneity of staff into an advantage. An extension for staff heterogeneity is discussed in Section 5.4.

3. Network flow model

As we investigate a shift rostering problem in which work rules are formulated in terms of the newly-proposed concept of prohibited meta-sequences, Section 3.1 presents an illustrative example that is used in the remainder of the paper. Section 3.2 discusses how work rules are modeled using both prohibited meta-sequences and resource constraints. Section 3.3 proposes a novel graph-based formulation for the traditional 3-index model *SRP*1.

Table 1
Example: demand requirements.

Shifts	Days					
	1	2	3	4	5	6
1 (morning = M)	39	83	30	33	36	66
2 (afternoon = A)	21	11	34	17	31	20
3 (night = N)	40	6	36	50	33	14
4 (day-off = D)	0	0	0	0	0	0

Table 2
Example: roster.

No. of staff	Shift pattern					
	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6
36	4	1	3	4	1	1
33	1	1	4	1	3	4
30	3	4	1	3	4	1
11	2	2	4	2	2	4
14	4	1	2	4	2	3
10	2	4	2	3	4	2
10	3	4	2	3	4	2
6	1	3	4	2	2	4
Total: 150						

3.1. An illustrative example

Consider the demand (number of staff required) for a 4-shift 6-day planning period (e.g. 17 people are required for afternoon shifts on day 4) in Table 1.

The objective of *SRP*1 is to schedule staff to cover the demand as much as possible such that all the work rules are observed.

The solution of an SRP is a roster showing the number of staff will be working on each shift during the planning horizon. A sequence of shift-assignments for the planning horizon is referred to as a shift pattern. Work rules are imposed to avoid the assignment of some shift patterns to the staff. The roster to cover the demand can be represented at an aggregate level by a set of feasible shift patterns that satisfy the work rules together with the number of staff assigned to each feasible shift pattern. Table 2 shows a roster that covers the demand requirements in Table 1. This roster uses 8 shift patterns, each represented by a row in the table, with the leftmost column indicating the number of staff that is assigned to the shift pattern (e.g. 36 staff members are scheduled for the sequence DMNDMM).

In *SRP*1, the staffing level for a shift is not required to match demand exactly, as both under-staffing and over-staffing are allowed. Over- and under-staffing penalty costs are to be minimized. All staff members are assumed to be subject to the same set of work rules, i.e. the staff is homogeneous.

3.2. Work rules

Many work rules can be easily formulated as resource constraints. Let $\mathcal{R}^k$ be the set of resources for staff member $k \in \mathcal{K}$. For each resource $r \in \mathcal{R}^k$ of staff member $k \in \mathcal{K}$, let $W_r^k \in \mathbb{Z}^+$ be the resource capacity and let $w_{ij}^{rk} \in \mathbb{Z}, 0 \le w_{ij}^{rk} \le W_r^k$, be the amount of resource r consumed when staff member k is assigned to cover shift $i \in \mathcal{I}$ on day $j \in \mathcal{J}$. The resource usage for each staff member must not exceed the resource capacity and the resource constraints can be formulated as

$$\sum_{i\in\mathcal{I}} \sum_{j\in\mathcal{J}} w_{ij}^{rk} x_{ij}^k \le W_r^k, \qquad \forall r \in \mathcal{R}^k, k \in \mathcal{K}. \tag{7}$$

Although resource constraints can model many work rules, some work rules can be more conveniently formulated by using the prohibited meta-sequences that we introduce in this

Table 3
Example: work rules as prohibited meta-sequences.

Work rules	Prohibited meta-sequences
a day-off should be assigned after a night shift.	({3}, {1, 2, 3})
no night-dayoff-night shift sequence.	({3}, {4}, {3})
at most 2 consecutive working shifts.	({1, 2, 3}, {1, 2, 3}, {1, 2, 3})
at least 2 consecutive working shifts.	({4}, {1, 2, 3}, {4})

paper. We define a prohibited meta-sequence m as a sequence of subsets of shifts $(S_m(1), S_m(2), \ldots, S_m(L_m))$ where $S_m(l) \subseteq \mathcal{I}$ for all $l \in \{1, 2, \ldots, L_m\}$. The number of components in a meta-sequence m, denoted by L_m, is referred to as the length of the meta-sequence. Let $\mathcal{S}^k$ represent the set of prohibited meta-sequences for staff member $k \in \mathcal{K}$. A prohibited meta-sequence $m \in \mathcal{S}^k$ disallows the assignments of the shift sequences in $S_m(1) \times S_m(2) \times \cdots \times S_m(L_m)$ to staff member k. The constraints enforcing the prohibited meta-sequences are formulated as

$$\sum_{l=1}^{L_m} \sum_{i \in S_m(l)} x^k_{i,j+l} \leq L_m - 1, \quad \forall j \in \{0, \ldots, J - L_m\}, m \in \mathcal{S}^k, k \in \mathcal{K}. \quad (8)$$

The work rules formulated in equations (6a)–(6d) can be naturally expressed by prohibited meta-sequence as shown in Table 3. As an example, the work rule *"at least 2 consecutive working shifts"* is formulated by prohibited meta-sequence ({4}, {1, 2, 3}, {4}) that disallows the shift sequences (4,1,4), (4,2,4) and (4,3,4). Some work rules, e.g. on minimum/maximum work days within the planning period, are easily modeled as resource constraints. Other work rules on disallowed sequences of shift types (e.g. no night-morning-night shift sequence) are best modeled as prohibited meta-sequence constraints.

Since we consider a shift rostering problem with homogeneous staff, all staff are subject to the same set of work rules and therefore we can set

$$\mathcal{S} = \mathcal{S}^k, \ \mathcal{R} = \mathcal{R}^k, \ w^r_{ij} = w^{rk}_{ij}, \ W_r = W^k_r \quad \forall k \in \mathcal{K}, i \in \mathcal{I}, j \in \mathcal{J}, r \in \mathcal{R}^k.$$

Although this paper focuses on a shift rostering problem with homogeneous staff, the model and the proposed algorithm are applicable for shift rostering problems with heterogeneous staff. An extension for staff heterogeneity is discussed in Section 5.4.

3.3. Network model

In this section, a graph-based formulation is proposed for which the size of the model depends on the structure of the work-rule constraints and is independent of the number of staff members. Given a set of resource constraints and a set of prohibited meta-sequences, Section 4 describes how a directed acyclic multigraph $G(V, E)$ is constructed with a source $s \in V$ and a sink $t \in V$ such that the set of (s, t)-paths corresponds to the set of feasible shift patterns. Each edge in E is associated with one or several assignments of shifts to days. Vertices in V correspond to the states of (s, t)-paths defined in Section 4.3 such that all (s, t)-paths in graph G meet all resource constraints and do not contain any prohibited meta-sequence. The assignments along an (s, t)-path therefore give a feasible shift pattern that can be assigned to any of the K staff members. Thus, a K-flow in the graph corresponds to a feasible solution to the SRP. Graph G is acyclic to avoid having (s, t)-paths that would traverse any arc more than once, implying the infeasible assignment of the same shift multiple times on the same day.

Fig. 1 illustrates the network graph that represents feasible shift patterns for the example introduced in Section 3.1. The edge from the source s to vertex 3 is associated with {(4, 1), (1, 2)} which corresponds to the assignment of shift 4 on day 1 and shift 1 on day 2. The edges of the path $(s, 3, 5, 7, t)$ are associated with the set of assignments {(4, 1), (1, 2)}, {(3, 3)}, {(4, 4), (1, 5)} and {(1, 6)}. This path therefore corresponds to the shift pattern (4, 1, 3, 4, 1, 1). Note that any (s, t)-path in the graph in Fig. 1 corresponds to a feasible shift pattern for the example. All the 120 feasible shift patterns are represented by the graph that consists of only 9 vertices and 28 edges. The example demonstrates the potential of representing a large number of feasible shift patterns using a graph of a small size.

Using the graph representation of the work rules which is further described in Section 4, *SRP*1 can be reformulated as a network flow model where the demand requirements are handled as side constraints. For all edges $e \in E$, shifts $i \in \mathcal{I}$ and days $j \in \mathcal{J}$, let

$$a^e_{ij} = \begin{cases} 1, & \text{if shift } i \text{ on day } j \text{ is associated with edge } e; \\ 0, & \text{otherwise.} \end{cases}$$

Let y_e be the flow on edge $e \in E$ corresponding to the number of staff assigned. Since K staff members need to be assigned to one of the shifts (which include the possibility of having a day-off), *SRP*2 then consists of finding a K-flow with minimal overall costs in the graph $G(V, E)$ with a source $s \in V$ and a sink $t \in V$ such that the set of (s, t)-paths corresponds to the set of feasible shift patterns. For all $v \in V$, let $\delta^-(v)$ and $\delta^+(v)$ be the subset of edges in E that are incident to and incident from vertex v, respectively.

$$(SRP2): \quad min \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} (\alpha_{ij} \, u_{ij} + \beta_{ij} \, o_{ij}) \quad (9)$$

$$s.t. \sum_{e \in \delta^-(v)} y_e - \sum_{e \in \delta^+(v)} y_e = 0, \qquad \forall v \in V \setminus \{s, t\}, \quad (10)$$

$$\sum_{e \in \delta^+(s)} y_e = K, \quad (11)$$

$$\sum_{e \in \delta^-(t)} y_e = K, \quad (12)$$

$$\sum_{e \in E} a^e_{ij} y_e + u_{ij} - o_{ij} = d_{ij}, \qquad \forall i \in \mathcal{I}, j \in \mathcal{J}, \quad (13)$$

$$u_{ij}, o_{ij} \geq 0, \qquad \forall i \in \mathcal{I}, j \in \mathcal{J}, \quad (14)$$

$$y_e \geq 0, integer, \qquad \forall e \in E. \quad (15)$$

Objective function (9) is the same as (1) in *SRP*1 and minimizes the sum of under- and over-staffing penalties. Constraints (10)–(12) ensure that the solution is a K-flow in graph G. The coverage constraints (13) consider the demand requirements for the shifts, where the number of staff assigned to shifts is indicated by the flow on the corresponding edge.

*SRP*2 can be solved using a standard commercial solver. The solution of *SRP*2 for the example problem is illustrated in Fig. 2, in which edges with no staff assigned are not depicted. The label on each edge indicates the flow value and its associated shift assignments. The edge incident from the source s to vertex 3 is associated with the assignment of shift 4 on day 1 and shift 1 on day 2, and its flow value of 50 indicates that exactly 50 staff members will get these shift assignments.

If one wants to determine the number of staff members that are to be assigned to each shift pattern, the (s, t)-paths can be determined by decomposing the K-flow solution of *SRP*2 with the following procedure.

Step 1. Remove arcs $e \in E$ with $y_e = 0$, and let $\hat{G}(\hat{V}, \hat{E})$ denote the resulting graph.
Step 2. For all $e \in \hat{E}$, associate a weight W_e on arc e, and initialize W_e to y_e.
Step 3. Pick an (s, t)-path P in graph $\hat{G}$ maximizing $\min_{e \in P} W_e$. Let P^* denote the selected path and define W^* as $\min_{e \in P^*} W_e$. To generate alternative solutions, randomly pick an (s, t)-path P in graph $\hat{G}$ with $W_e > 0$ for all $e \in P$.

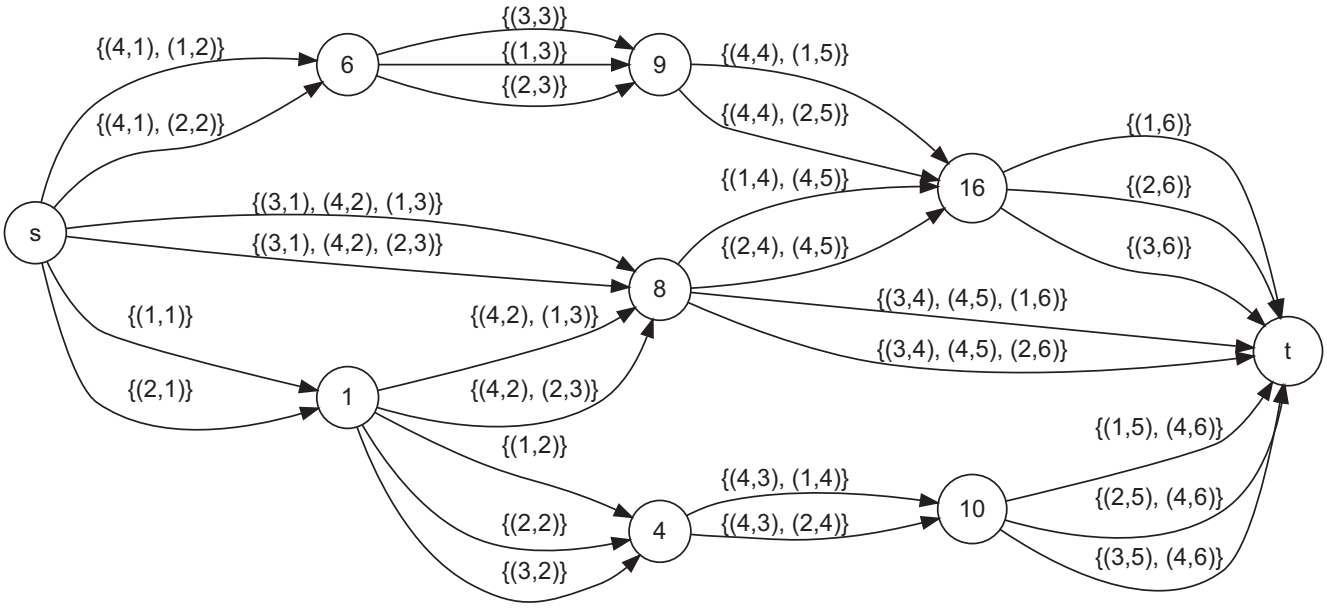

Fig. 1. Example: the underlying graph with feasible shift patterns.

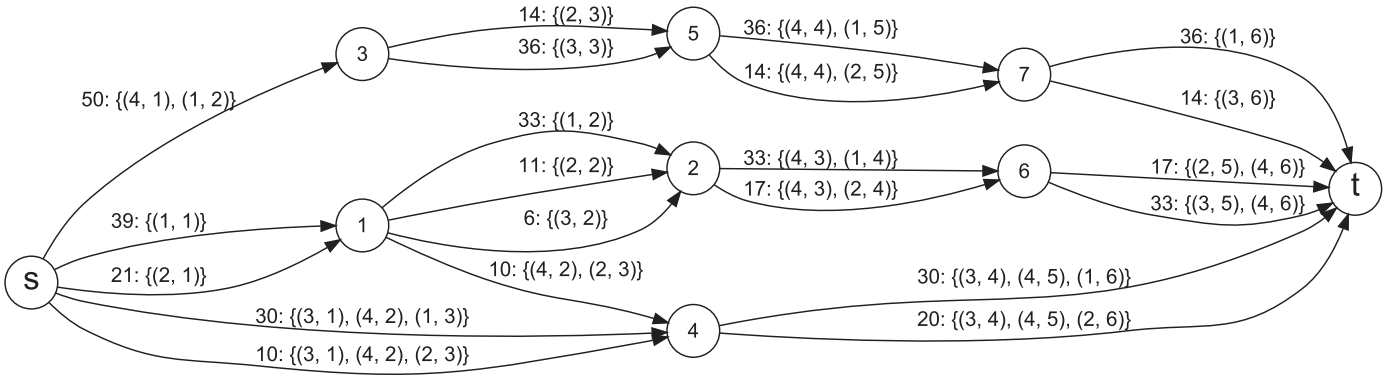

Fig. 2. Example: a feasible K-flow solution.

Step 4. Record the shift pattern represented by P and assign the shift pattern to W^* number of workers.
Step 5. Set W_e to $W_e - W^*$ for all $e \in P$.
Step 6. Remove the arcs $e \in \hat{E}$ with $W_e = 0$.
Step 7. Repeat steps 3 to 6, and terminate the algorithm when all the K workers are assigned with a shift pattern.

The algorithm described above runs in polynomial-time on the size of graph G. For each iteration (steps 3 to 6), an (s, t)-path in $\hat{G}$ can be determined in a polynomial time using a shortest path algorithm. For each iteration (steps 3 to 6), at least one worker is assigned to a shift pattern and at least one arc in the selected (s, t)-path P is removed from graph $\hat{G}$. Hence, the number of iterations the algorithm runs is bounded above by K and $|E|$.

The K-flow solution in Fig. 2 can be decomposed into the path-based solution shown in Table 4. Each row of the table shows an (s, t)-path that is assigned to the number of staff indicated in the left-most column. The corresponding vertices and shift pattern of the path are shown in the second and third column respectively. As shown in Table 4, an alternative path-based solution can be obtained from the graph in Fig. 2. In fact, a K-flow solution can obtain multiple alternative optimal solutions which are beneficial for managerial decisions in practice.

If all the work rules of $SRP1$ are expressible in the form of resource constraints (7) or prohibited meta-sequences (8), then $SRP1$ is equivalent to $SRP2$ on a network constructed according to Section 4 (where all path flows comply with the work rules constraints). Every K-flow can be decomposed into (s, t)-paths corresponding to shift assignments (i.e. x_{ij}^k values). Correspondingly, every solution of $SRP1$ corresponds to an optimal K-flow for $SRP2$ for the associated network.

In the next section, the graph is further discussed and an algorithm is proposed to construct such a graph from the work rules.

4. Graph construction algorithm

This section details how to construct a directed acyclic graph representing the set of all feasible shift patterns.

We first illustrate with a graph $G'(V, E)$ representing all possible shift patterns without any additional constraints. The vertex set $V = \{v_0, v_1, \ldots, v_J\}$ where $\mathcal{J} = \{1, 2, \ldots, J\}$ is the set of days in the planning horizon. Vetex $s = v_0$ is the source and vertex v_J is the sink. Shift i on day j is represented by an edge in E that incident from v_{j-1} to v_j. A path from s to t represents a shift pattern. Fig. 3 shows the graph representing all possible shift patterns for the example introduced in Section 3.1. A path from $s = \mathcal{N}_0$ to $t = \mathcal{N}_J$ represents a shift pattern with exactly one shift assigned in each day, starting from day 1 to day J.

Among the shift patterns represented by the graph, some may violate work rules. For example, shift pattern (1,1,1,1,1,1)

Table 4
Example: two solutions to the example problem.

No. of staff	Vertices of the path	Shift pattern	No. of staff	Vertices of the path	Shift pattern
36	(*s*, 3, 5, 7, *t*)	(4,1,3,4,1,1)	30	(*s*, 3, 5, 7, *t*)	(4,1,3,4,1,1)
33	(*s*, 1, 2, 6, *t*)	(1,1,4,1,3,4)	6	(*s*, 3, 5, 7, *t*)	(4,1,2,4,1,1)
30	(*s*, 4, *t*)	(3,4,1,3,4,1)	8	(*s*, 3, 5, 7, *t*)	(4,1,2,4,2,3)
11	(*s*, 1, 2, 6, *t*)	(2,2,4,2,2,4)	6	(*s*, 3, 5, 7, *t*)	(4,1,3,4,2,3)
14	(*s*, 3, 5, 7, *t*)	(4,1,2,4,2,3)	33	(*s*, 1, 2, 6, *t*)	(1, 1, 4, 1, 3, 4)
10	(*s*, 1, 4, *t*)	(2,4,2,3,4,2)	30	(*s*, 4, *t*)	(3, 4, 1, 3, 4, 1)
10	(*s*, 4, *t*)	(3,4,2,3,4,2)	11	(*s*, 1, 2, 6, *t*)	(2, 2, 4, 2, 2, 4)
6	(*s*, 1, 2, 6, *t*)	(1,3,4,2,2,4)	10	(*s*, 1, 4, *t*)	(2, 4, 2, 3, 4, 2)
			10	(*s*, 4, *t*)	(3, 4, 2, 3, 4, 2)
			6	(*s*, 1, 2, 6, *t*)	(1, 3, 4, 2, 2, 4)

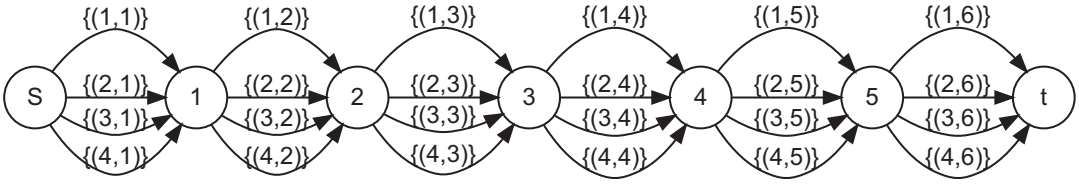

Fig. 3. Example: the graph with all shift patterns.

which specifies that shift 1 is performed each day, violates the example work rule that at most 2 consecutive working shifts can be performed. To eliminate the infeasible shift patterns from the graph, an algorithm is proposed that effectively splits the vertices of G', so that the resulting graph represents only the feasible shift patterns (Fig. 1). The algorithm relies on an efficient way to determine the feasibility of an (s, t)-path with respect to the resource constraints and the prohibited meta-sequences. We first describe how feasibility for one (s, t)-path is determined before introducing the proposed algorithm to construct the graph.

For every edge e, let $\sigma(e) \in \mathcal{I} \times \mathcal{J}$ denote the shift assignment associated with edge e. A *partial path* in G' is a sequence of edges forming a directed path that starts from source s. For any partial path $P = (p_1, p_2, \ldots, p_n)$, it is said to be *extended along* edge e *to* partial path Q if $Q = (p_1, p_2, \ldots, p_n, e)$.

4.1. Resource constraints

The feasibility of an (s, t)-path in graph G' with respect to the resource constraints may be determined as follows. For every edge $e \in \mathcal{A}$, the *resource consumption* of resource r on edge e is defined as

$$w_r(e) = \{w_{ij}^r : (i, j) = \sigma(e)\}, \qquad \forall r \in \mathcal{R}. \tag{16}$$

The *resource usage* of a partial path P in graph G' is defined as

$$c_r(P) = \sum_{e \in P} w_r(e), \qquad \forall r \in \mathcal{R}. \tag{17}$$

As the resource consumptions on all the edges are non-negative and additive, for any partial path $P = (p_1, p_2, \ldots, p_n)$ that is extended along edge e, the resource usage of the extended partial path $Q = (p_1, p_2, \ldots, p_n, e)$ can be computed from the resource usage of P and the resource consumption on edge e as follows.

$$c_r(Q) = c_r(P) + w_r(e), \qquad \forall r \in \mathcal{R}. \tag{18}$$

The feasibility of an (s, t)-path with respect to the resource constraints can therefore be determined from the resource usages of its partial paths which can be computed iteratively using (18).

4.2. Prohibited meta-sequences

The feasibility of an (s, t)-path in graph G' with respect to the prohibited meta-sequences may be determined as follows. For every edge $e \in E'$ that is associated with assignment $(i, j) \in \mathcal{I} \times \mathcal{J}$, let $\sigma_{\mathcal{I}}(e) = i$ be the shift associated with edge e. For any partial path $P = (p_1, p_2, \ldots, p_n)$ and meta-sequence $(S(1), S(2), \ldots, S(L))$ where $S(i) \subseteq \mathcal{I}$ for all $i \in \{1, 2, \ldots, L\}$, partial path P is said to *end with* the meta-sequence if

(i) $L \leq n$ and
(ii) $\sigma_{\mathcal{I}}(p_{n-L+i}) \in S(i), \ \forall i \in \{1, 2, \ldots, L\}$.

Note that a partial path ends with a meta-sequence if (i) the meta-sequence is not longer than the corresponding shift pattern, and if (ii) the ending part of the corresponding shift pattern matches the meta-sequence. For any partial path P in graph G' and any prohibited meta-sequence $(S_m(1), S_m(2), \ldots, S_m(L_m))$, and for $l \in \{1, 2, \ldots, L_m\}$, we define

$$h_l^m(P) = \begin{cases} 1, & \text{if } P \text{ ends with } (S_m(1), S_m(2), \ldots, S_m(l)); \\ 0, & \text{otherwise.} \end{cases} \tag{19}$$

to indicate if the last l edges of partial path P match the first l elements of meta-sequence m. It follows that an (s, t)-path corresponds to a shift pattern that is feasible with respect to the prohibited meta-sequence $m \in \mathcal{S}$ if and only if the path consists of no partial path P where

$$h_{L_m}^m(P) = 1. \tag{20}$$

For any partial path P in graph G' and prohibited meta-sequence $m \in \mathcal{S}$, we define the vector

$$H_m(P) = (h_1^m(P), h_2^m(P), \ldots, h_{L_m}^m(P))$$

as the *end-with vector* of P with respect to m, to track the overlap of P with the prohibited meta-sequence m. The feasibility of an (s, t)-path with respect to the prohibited meta-sequence m can be determined by the corresponding end-with vectors of its partial paths.

The end-with vector of a partial path can be easily computed when the partial path is extended. For $m \in \mathcal{S}$ and $e \in \mathcal{A}$, let $b_m(e) = (b_1^m(e), b_2^m(e), \ldots, b_{L_m}^m(e))$ be a binary vector where

$$b_l^m(e) = \begin{cases} 1, & \text{if } \sigma_{\mathcal{I}}(e) \in S_m(l); \\ 0, & \text{otherwise,} \end{cases} \qquad \forall l \in \{1, 2, \ldots, L_m\}, \tag{21}$$

which we call the *inclusion vector* of edge e with respect to meta-sequence m. For any partial path P in graph G', we set

$$h_0^m(P) = 1, \qquad \forall m \in \mathcal{S}. \tag{22}$$

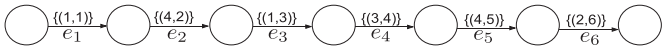

Fig. 4. Example path with shift assignments associated on edges.

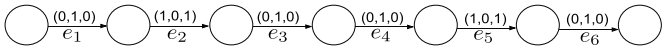

Fig. 5. Example path with inclusion vectors associated on edges.

Fig. 6. Example end-with vectors with respect to the extended partial paths.

Then, for any partial path $P = (p_1, p_2, \ldots, p_n)$ that is extended along edge e, the end-with vector of the extended partial path $Q = (p_1, p_2, \ldots, p_n, e)$ with respect to meta-sequence m can be computed as follows.

$$h_l^m(Q) = h_{l-1}^m(P) b_l^m(e), \qquad \forall l \in \{1, 2, \ldots, L_m\}, m \in \mathcal{S}. \tag{23}$$

For notational simplicity, we define an operator $\star : \mathbb{B}^n \times \mathbb{B}^n \mapsto \mathbb{B}^n$ on two n-dimensional vectors x and y, as follows:

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_n \end{pmatrix} \star \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} y_1 \\ x_1 y_2 \\ x_2 y_3 \\ \vdots \\ x_{n-1} y_n \end{pmatrix}.$$

Then (23) can be rewritten as (24).

$$H_m(Q) = H_m(P) \star b_m(e), \qquad \forall m \in \mathcal{S}. \tag{24}$$

As an illustration, consider the prohibited meta-sequence ({4}, {1, 2, 3}, {4}) corresponding to the side constraint of having at least 2 consecutive working shifts (see Table 3) and (s, t)-path $(e_1, e_2, e_3, e_4, e_5, e_6)$ where the edges are associated with assignments {(1, 1)}, {(4, 2)}, {(1, 3)}, {(3, 4)}, {(4, 5)} and {(2, 6)}, respectively, as displayed in Fig. 1. The path is illustrated in Fig. 4.

The inclusion vectors of the edges $(e_1, \ldots, e_6)$ with respect to $m' = (\{4\}, \{1, 2, 3\}, \{4\})$, $b_{m'}(e)$, are obtained using (). The inclusion vectors are indicated in Fig. 5 on the corresponding edges. For example, edge e_2, which assigns shift 4 on day 2, has an inclusion vector (1, 0, 1) since shift 4 belongs to the first and the third components of ({4}, {1, 2, 3}, {4}), but not to the second component of the prohibited meta-sequence.

To determine the feasibility of the path, we start with an empty partial path with an end-with vector (0,0,0), and then extend the partial paths iteratively. The end-with vectors with respect to m of the extended partial paths are obtained using (24) and are illustrated in Fig. 6.

For example, partial path $(e_1, \ldots, e_4)$ has end-with vector (0,0,0) with respect to meta-sequence m'. When $(e_1, \ldots, e_4)$ is extended along edge e_5 to partial path $(e_1, \ldots, e_5)$, the end-with vector of $(e_1, \ldots, e_5)$ with respect to m' is equal to $(0, 0, 0) \star (1, 0, 1) = (1, 0, 0)$. As the last digit of the end-with vector is zero, by (20), the path corresponding to the shift pattern (1,4,1,3,4,2) is feasible with respect to the prohibited meta-sequence ({4}, {1, 2, 3}, {4}).

4.3. Algorithm for constructing the graph of feasible shift patterns

We construct a graph $G(V, E)$ to represent the set of feasible shift patterns, where the vertices in V represent states of feasible partial paths. A partial path P is said to be *feasible* when

$$c_r(P) \le W_r, \quad \forall r \in \mathcal{R}, m \in \mathcal{S}, \text{ and } h_{L_m}^m(P) = 0. \tag{25}$$

Otherwise, the partial path is said to be *infeasible*. It follows that an (s, t)-path corresponds to a feasible shift pattern if and only if it includes no infeasible partial paths.

To determine the feasibility of a partial path, we define the *state* of a partial path $P = (p_1, p_2, \ldots, p_n)$ as follows.

$$[d(P), \{c_r(P) : r \in \mathcal{R}\}, \{H_m(P) : m \in \mathcal{S}\}]$$

where $d(P) = n \in \mathcal{N}$ denotes the number of shift assignments, $c_r(P)$ denotes the resource usage and $H_m(P)$ is the end-with vector of the partial path. If partial paths of the same state extend along the same edge, the states of the extended partial paths are identical. In our graph representation G, each vertex is associated with a state. If two partial paths have the same state, then they both end at the same vertex in the graph G. Potentially, the number of states (hence vertices) could be as many as $|J| \prod_{r\in\mathcal{R}} W_r \prod_{m\in\mathcal{S}} L_m$ in which $|J|$ is the number of days in the planning horizon, $\mathcal{R}$ is the set of resources and $\mathcal{S}$ the set of meta-sequences. As a result, the number of states is exponential in $|\mathcal{R}|$ and $|\mathcal{S}|$. However, since infeasible paths are pruned during network construction, one can expect network size to grow at a much lower rate as will be examined in Section 5.

The graph is constructed iteratively from the source $s = v_0$ by considering extension along an edge $(i, j) \in \mathcal{I} \times \mathcal{J}$. We start with an empty partial path with state $v_0 \in V$ (representing the beginning of the planning horizon) where

$$\begin{aligned} d(v_0) &= 0, \\ c_r(v_0) &= 0, \quad \forall r \in \mathcal{R}, \\ H_m(v_0) &= \mathbf{0}, \quad \forall m \in \mathcal{S}. \end{aligned}$$

At each iteration, a feasible partial path is extended along an edge (i, j) corresponding to a shift assignment for the next day. If the feasible partial paths corresponding to state $u \in V$ can be extended along edge e to a feasible partial path of state $v \in V$, then an edge is introduced in E, incident from u to v which is associated with assignments $\sigma(e)$. The state of the extended partial path can be computed as follows.

$$\begin{aligned} d(v) &= d(u) + 1, \\ c_r(v) &= c_r(u) + w_r(e), \quad \forall r \in \mathcal{R}, \\ H_m(v) &= H_m(u) \star b_m(e), \quad \forall m \in \mathcal{S}. \end{aligned}$$

Note that the paths can be extended along e only if $\sigma(e) = (i, j)$ with $j = d(u) + 1$. The extended partial path is feasible when (25) holds.

It takes $O(|\mathcal{R}| + \sum_{m\in\mathcal{S}} L_m)$ time to compute state v and to determine feasibility. Observing that feasible partial paths of the same state could be consolidated to end at the same vertex in V, a new vertex is introduced in V only when the state of the extended partial path has not been introduced in the previous iterations. To implement the algorithm, we recommend using a hash function to create key values for the states in V and store the keys that map to the states in a one-to-one correspondence. This enables individual states in V to be retrieved based on their keys in $O(|V|)$ time.

We define s as the source and $\{v \in V : d(v) = J\}$ as the sinks of graph G. For any path in graph G, from the source to a sink, the feasibility of its partial paths are indicated by the states of the corresponding vertices along the path. Since we extend partial paths only when feasible, any path from a source to a sink corresponds to a feasible shift pattern. By searching for all extensions from all the feasible vertices, graph G represents the set of all feasible shift patterns. As there are at most $|\mathcal{I}|$ number of out-going edges for any vertex, the algorithm runs with at most $|\mathcal{I}||V|$ iterations. Therefore, the algorithm runs in $O(|\mathcal{I}||V||\mathcal{R}| + |\mathcal{I}||V| \sum_{m\in\mathcal{S}} L_m + |\mathcal{I}||V|^2)$ time. Note that the running time is bounded above by a polynomial in the size of the graph, the number of resource constraints and the total length of all the meta-sequences. If work

rules occurring in practice are used, the graph sizes are often manageable as discussed in Section 5.

An optimal K-flow in the graph can be decomposed into K (s, t)-paths in $O(KJ)$ time where K is the number of staff and J is the number of days. The (s, t)-paths can then be transformed into K feasible paths to find an optimal solution to the shift rostering problem. As the same K-flow can usually be decomposed into many different sets of (s, t)-paths, the formulation *SRP*2 yields multiple optimal solutions to the problem as already mentioned in Section 3.3.

4.4. Graph Simplifications

The algorithm proposed for constructing the graph of feasible shift patterns can be considered to be efficient for graphs of small size based on the algorithm's worst case complexity. It can be shown that prohibited meta-sequences with the following characteristics lead to relatively small graphs if:

(i) The length L of a prohibited meta-sequence is short;
(ii) All components in a prohibited meta-sequence ($S(1)$, $S(2)$, ..., $S(L)$) are disjoint. i.e. For all $i, j \in \{1, 2, ..., L\}$ where $i \neq j$, $S(i) \cap S(j) = \emptyset$;
(iii) Prohibited meta-sequences $(S(1), S(2), \ldots, S(L))$ have a pyramidal structure in which $S(1) \subseteq S(2) \subseteq \cdots \subseteq S(L)$ or $S(1) \supseteq S(2) \supseteq \cdots \supseteq S(L)$.

In all cases, it is worthwile to try simplifying the graph using two operations: *merge* and *contract*.

As an example, consider the graph shown in Fig. 7 and notice how the subgraphs of path extensions are very similar on the edges near vertex t. Some of the duplications may be consolidated as shown in Fig. 8. The subgraph shown on the left can be replaced by the subgraph shown on the right with one set of the duplicated edges (1,5) and (2,5) removed. This *merge* operation preserves the set of feasible shift patterns because the partial paths that end at vertex 24 and the partial paths that end at vertex 25 would be extended along edges with the same set of shift assignments to the same vertex 34 and henceforth to the same subsequent vertices.

To formally define the merge operator, let edge $e \in E$ be incident from $\mathcal{T}(e) \in V$ to $\mathcal{H}(e) \in V$ and be associated with assignments $\sigma(e) \subseteq (\mathcal{I} \times \mathcal{J})$. For any vertex $v \in V$, let $\delta_-(v) \subseteq E$ and $\delta_+(v) \subseteq E$ denote the set of incoming edges and out-going edges of vertex v respectively. For any vertex-pair $(v_1, v_2) \in V \times V$ where $v_1 \neq v_2$, we say $\delta_+(v_1) \equiv \delta_+(v_2)$ if and only if for every edge $e_1 \in \delta_+(v_1)$, there is an edge $e_2 \in \delta_+(v_2)$ such that $\sigma(e_1) = \sigma(e_2), \mathcal{H}(e_1) = \mathcal{H}(e_2)$ and vice versa. If $\delta_+(v_1) \equiv \delta_+(v_2)$, then v_1 and v_2 are merged into a single vertex as follows. For all edges $e \in \delta_-(v_1)$, an edge incident from vertex $\mathcal{T}(e)$ to vertex v_2 is introduced, which is associated with assignments $\sigma(e)$. Vertex v_1 and all edges in $\delta_-(v_1)$ and $\delta_+(v_1)$ are removed. Essentially, the merge operator is only applied when partial paths ending on two vertices can be extended along edges that are associated with the same shift assignments to the same subsequent vertices.

The merge operator always preserves the set of feasible shift patterns the graph represents. Let P_1 and P_2 be the edges that are incident to vertex v_1 and v_2 respectively. Let Q_1 and Q_2 be the edges that incident from vertex v_1 and v_2, respectively. The operation removes (s, t)-paths (..., e_1, e_2, ...) for all $e_1 \in P_1$ and $e_2 \in Q_1$ (that traverses v_1) and introduces (s, t)-paths (..., $\bar{e}_1, \bar{e}_2$, ...) (that traverses v_2) where $\bar{e}_1 = (\mathcal{T}(e_1), v_2)$ and $e_2 \in Q_2$ with $\sigma(e_1) = \sigma(\bar{e}_1), \sigma(e_2) = \sigma(\bar{e}_2)$ and $\mathcal{H}(\bar{e}_2) = \mathcal{H}(e_2)$. These operations preserve the set of feasible paths the graph represents, since the set of shift assignments associated with the paths that are removed is equivalent to the ones associated with the paths that are introduced. The merge operator is applied repeatedly until the graph cannot be further simplified. The resulting graph for the example problem at this point is illustrated in Fig. 9.

After applying the merge operations, the graph can be further simplified by contracting simple paths as illustrated in Fig. 10. Vertex $v \in V$ is contracted as follows. For each edge-pair $(e_1, e_2) \in \delta_-(v) \times \delta_+(v)$, an edge that incidents from $\mathcal{T}(e_1)$ to $\mathcal{H}(e_2)$ is introduced, which is associated with assignments $\sigma(e_1) \cup \sigma(e_2)$. Vertex v and all edges in $\delta_-(v)$ and $\delta_+(v)$ are removed. Essentially, the sub-paths containing $(e_1, e_2) \in \delta_-(v) \times \delta_+(v)$ are removed, and replaced by an edge e that represents the same set of shift assignments. Thus, the set of feasible shift patterns is preserved. To ensure that the operation does not increase the number of edges, we contract a vertex $v \in V$ only if $|\delta_+(v)||\delta_-(v)| \leq |\delta_-(v)| + |\delta_+(v)|$, which holds when either $|\delta_-(v)| = 1$, or $|\delta_+(v)| = 1$, or $|\delta_+(v)|$ and $|\delta_-(v)|$ are both less than or equal to 2. The resulting graph for the example problem after merging and contracting is shown in Fig. 1.

Since a merge operation or a contract operation reduces the number of vertices by one, the total number of operations needed is not larger than the number of vertices.

After the graph generation and simplification procedures, we obtain a graph where all the feasible shift patterns are represented. Thus, solving *SRP*2, which is a network flow problem with a side constraint, on this graph would yield a globally optimal solution for the *SRP*.

5. Computational results

In this section, we compare the performance of the traditional 3-index formulation for the shift rostering problem (*SRP*1) to the newly proposed graph-based formulation (*SRP*2) when using standard commercial solvers. These problem formulations consist of hard constraints and a single soft constraint on the coverage for a homogeneous workforce. Although Doi et al. (2018) pointed out that in practice constraints are typically hard, the majority of the literature has focused on solving extended rostering problems involving additional soft constraints resulting in several benchmark instances being available for computational testing (see e.g. benchmark instances from competitions PATAT2010[1] and PATAT2016[2]). Rostering problems with hard constraints have been solved in recent literature as a part of a hybrid solution approach (see e.g. Doi et al., 2018; Rahimian et al., 2017) but the (real-life) problem instances have either not being disclosed or a comparison of results is impossible as the workforce is not considered to be homogeneous due to shift preferences for each staff member.

Therefore, instead of removing the soft constraints from the existing benchmarks instances we have decided to construct a benchmark set to handle the different work rules that Musliu (2006) identified appearing in practice in different industries such as airports, factories, health care organizations, etc. Our benchmark set consists of 500 instances: the 20 instances that Musliu (2006) created based on real-life problems (set 1); 20 additional medium-sized and 460 additional large-sized problem instances with the same work-rule constraints (set 2). To test our graph construction algorithm with different work rules settings we have additionaly generated 80 instances with random work rules (set 3). Details on the work rules generation are introduced in Section 5.3. The complexity of the problem instances increases from dataset 1 to 3. The two formulations are compared on dataset 1 (in experiment 1, Section 5.1) and on the medium-sized instances of dataset 2 (in experiment 2, Section 5.2). The classical 3-index formulation (*SRP*1) is not able to handle the large-sized instances

[1] https://www.kuleuven-kulak.be/nrpcompetition.
[2] http://mobiz.vives.be/inrc2/.

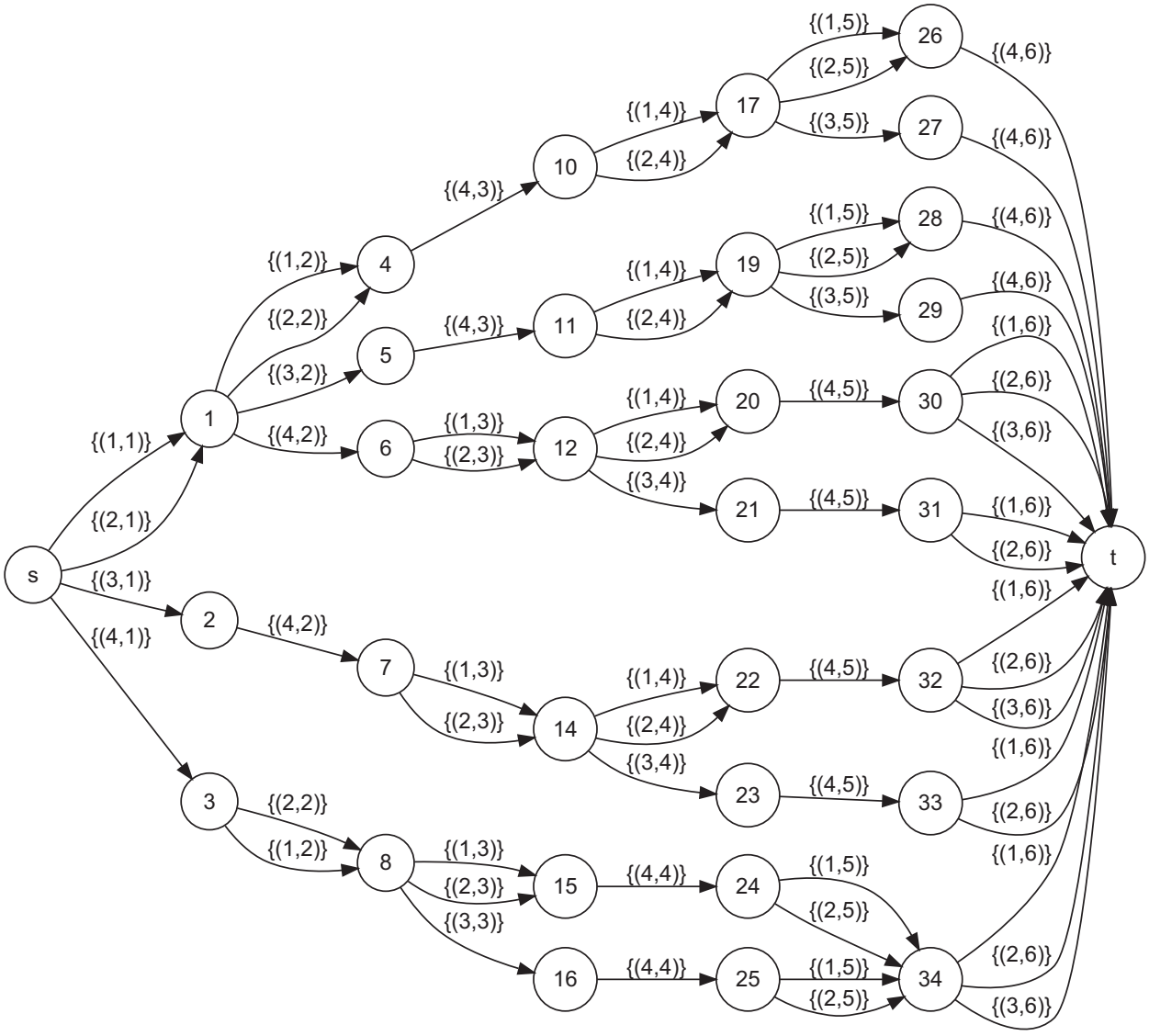

Fig. 7. Example: the graph before simplifications.

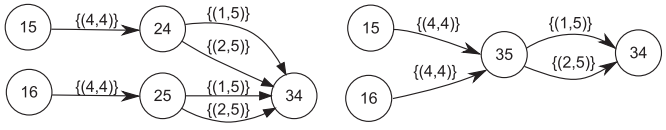

Fig. 8. Example: *merge* operation. Merging vertex 24 and vertex 25 of the graph on the left results in the graph shown on the right.

of dataset 2 and the instances of dataset 3 (in experiment 3, Section 5.3). To test the potential of the graph-based model and of the graph construction algorithm for related rostering problems with heterogeneous staff, an extensive benchmark dataset 4 with different problem settings was used (in experiment 4, Section 5.4). Datasets 2 and 3 and all the solutions files can be found in the associated supplementary material.

We identify the instances as Dμ_1-Sμ_2 – μ_3 where μ_1 represents the number of days in the planning period (J), μ_2 is the staffing ratio and μ_3 denotes the set of work rules. The average number of staff required per day is given by $\lambda = \sum_{i\in\mathcal{I}, j\in\mathcal{J}} \frac{d_{ij}}{|\mathcal{J}|}$, where d_{ij} is the demand in terms of the number of staff required. The staffing ratio of an instance, μ_2, is defined such that the number of staff to be scheduled is set to the nearest integer of $\mu_2\lambda$, i.e. $\mu_2 = \frac{K}{\lambda}$ where K is the number of staff. The set of work rules proposed by Musliu (2006) are denoted by P01-P20. For example, instance D30-S1.4-P01 is a 30-day rostering problem with a 1.4 staffing ratio and work rule set P01. To penalize over- and under-staffing, the values of α and β in the objective function are set to 1 for dataset 1, and to a random integer between 1 to 10 for datasets 2 and 3. Since a day-off is modeled as a dummy shift in the formulation, it is not related to over- or under-staffing, and thus α and β in the objective function are set to 0 for all shifts that represent day-offs.

A summary of the instances of each experiment set is presented in Table 5 and the instances and detailed descriptions are available upon request.

Experiments 1–3 were conducted on a personal computer running Windows 7 with an Intel Core i5-4570 processor 3.2 GHz and 4 GB of main memory. All algorithms are implemented in C++ and have been compiled using Visual Studio 2017. The solver Gurobi 7.5.2 with default settings was used to solve models *SRP*1 and *SRP*2 and their LP relaxations. Experiment 4 was conducted on a personal computer running Ubuntu with an Intel Core i5-7400 processor 3.00 GHz and 8 GB of main memory. The algorithms

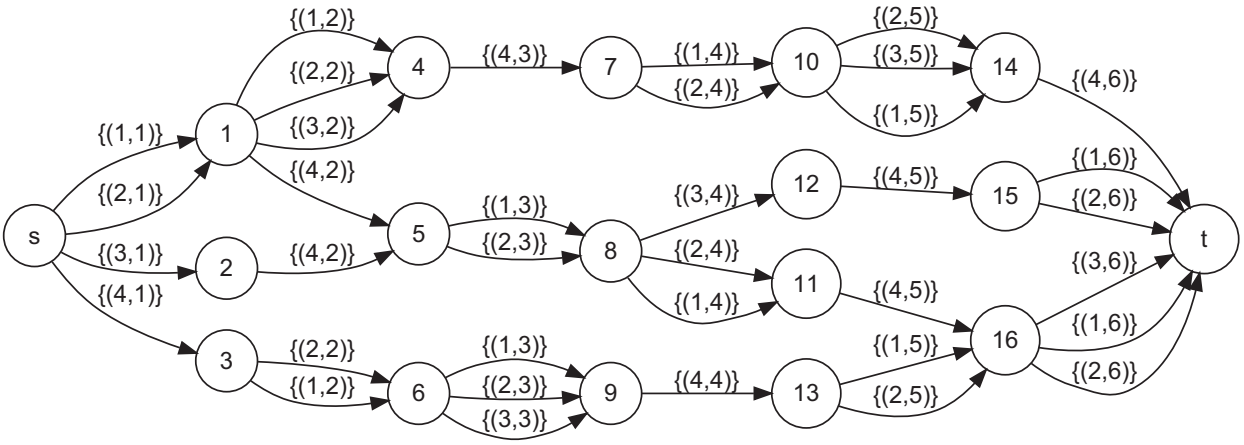

Fig. 9. Example: the graph after merging. The vertices are renumbered for ease of illustration.

Table 5
Summary of the problem instances.

Data sets	μ_1	μ_2	μ_3	I	K	nInst
Set 1	30	1.4	P01-P20	3/4	7/163/36.7	20
Set 2	30/60/90/120/150/180	1.2/1.4/1.6/1.8	P01-P20	10	495/877/674.8	480
Set 3	30	1.4	random	10	580/675/630.7	80
Set 4	7/28	2-5	Cases 1-16	4	25/30/50/60/75/100	248640

I = No. of shifts; K = No. of available staff (minimum/maximum/average); nInst = No. of instances.

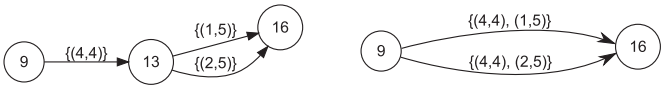

Fig. 10. Example: contract operation. By contracting vertex 13 in the graph on the left results in the graph shown on the right.

are implemented in C++ and have been compiled using GCC 9.2. The solver CPLEX V12.7.0 with default settings was used to solve models *SRP*1 and *SRP*2 and their LP relaxations.

Sections 5.1 to 5.4 discuss the results for experiments 1 to 4, respectively.

5.1. Experiment 1

In this section, the two formulations are compared on the 20 instances created by Musliu (2006) based on real-life problems. The original instances rely on a cyclic schedule of 7 days. To create instances with an acyclic planning horizon of 30 days, the original weekly demand pattern from Musliu (2006) is repeated four times. To complete the 30-day schedule, the demands of the first and second day of the weekly demand pattern are used. In practice, there may be assignments (working shifts or days-off) before or after the planning horizon, but we do not consider them, as we only consider constraints within the current planning horizon.

The results are presented in Table 6. The first column identifies the instances according to the number of days in the planning period, staffing ratio and work-rule scenario. The number of shifts and the total number of staff available are reported in columns 2 and 3, respectively. Columns 4 to 6 show the results of solving *SRP*1 ((1)–(5), (7)–(8)): the number of binary variables, the number of constraints, and the CPU time (in seconds), respectively. Columns 7 to 12 report the results of solving *SRP*2: the number of variables, the number of constraints, the CPU times (in seconds) for constructing the graph, simplifying the graph, and solving the graph-based model with the Gurobi MIP solver, and the total CPU time (in seconds). A speedup factor of the new approach in relation to the traditional formulation is shown in the right-most column. Both *SRP*1 and *SRP*2 are tested with each instance under the same circumstances including the work-rule constraints (denoted by the last component in the name of each instance).

The computational results on these benchmark instances show that all the 20 instances can be solved more efficiently using the graph-based formulation, and especially the instances with more staff. Overall average speedup factor of 186.83 is achieved with individual problem speed factors ranging from 3.56 for small models (approximately 1000 variables and 4000 constraints in *SPR*1) to several hundreds for larger problem formulations. Speedup factors do differ depending on the number of staff to be allocated (K) and the specific work rules involved.

5.2. Experiment 2

Experiment 2 compares the performance of the two model formulations on 480 randomly generated problem instances with the same work-rule constraints as in experiment 1 (proposed by Musliu, 2006). These are 20 medium-sized instances tested with both *SRP*1 and *SRP*2; the remaining 460 large-sized instances are tested with only *SRP*2.

5.2.1. *Comparing* SRP*1 and* SRP*2 model formulations*

The results on the 20 medium-sized instances with a 30-day planning horizon, 1.4 staffing ratio and the 20 sets of practical work rules P01-P20 are summarized in Table 7. Each row of the table shows the number of available staff (K) and the performance of both models in each instance within a time limit of 2 hours: number of variables and constraints; the best lower and upper bounds obtained by *SRP*1 (LB and UB, respectively); the optimal value; and the computational time (in seconds). A speedup factor is also reported in the right-most column.

As shown in Table 7, *SRP*1 has again a much higher dimension than *SRP*2 as it involves substantially more variables and constraints. As a result, *SRP*1 is not able to obtain an optimal solution for all the medium-sized instances, and a feasible solution can be obtained within the 2-hours time limit for only 6 instances. As shown in the right-most column, *SRP*2 can be

Table 6
Results of *SRP*1 and *SRP*2 on instances of set 1.

			3-index formulation (*SRP*1)			Graph-based formulation (*SRP*2)						
Instance (1)	*I* (2)	*K* (3)	No. of Binary Variables (4)	No. of Constraints (5)	Time(s) (6)	No. of Variables (7)	No. of Constraints (8)	Time(s) Construction (9)	Time(s) Simplification (10)	Time(s) MIP (11)	Time(s) total (12)	Speedup factor (6)/(12)
D30-S1.4-P01	4	9	1080	4008	1.139	1248	276	0.055	0.037	0.145	0.320	3.56
D30-S1.4-P02	4	9	1080	5403	3.621	800	220	0.049	0.106	0.044	0.230	15.74
D30-S1.4-P03	4	17	2040	7447	9.257	1296	275	0.054	0.041	0.080	0.222	41.70
D30-S1.4-P04	4	13	1560	6503	1.02	1456	327	0.062	0.037	0.031	0.163	6.26
D30-S1.4-P05	4	11	1320	5818	13.575	1389	323	0.060	0.035	0.084	0.216	62.85
D30-S1.4-P06	4	7	840	3725	0.914	1454	323	0.065	0.041	0.045	0.182	5.02
D30-S1.4-P07	4	29	3480	12619	237.559	1296	275	0.053	0.040	0.211	0.352	674.88
D30-S1.4-P08	4	16	1920	6600	5.216	1338	277	0.051	0.041	0.074	0.198	26.34
D30-S1.4-P09	4	47	5640	17792	6.418	1336	339	0.052	0.043	0.089	0.222	28.91
D30-S1.4-P10	4	27	3240	11757	5.072	1296	275	0.053	0.040	0.267	0.400	12.68
D30-S1.4-P11	4	30	3600	12360	34.481	1256	275	0.047	0.034	0.128	0.246	140.17
D30-S1.4-P12	3	20	1800	6530	12.271	654	169	0.022	0.010	0.038	0.098	125.21
D30-S1.4-P13	4	24	2880	9240	3.662	1305	322	0.046	0.030	0.132	0.243	15.07
D30-S1.4-P14	4	13	1560	6854	16.836	1389	323	0.060	0.035	0.112	0.256	65.77
D30-S1.4-P15	4	64	7680	31544	298.615	1266	326	0.051	0.027	0.093	0.223	1339.08
D30-S1.4-P16	4	29	3480	12706	10.288	1208	274	0.048	0.033	0.171	0.289	35.60
D30-S1.4-P17	3	33	2970	9858	1.718	678	171	0.020	0.012	0.039	0.099	17.35
D30-S1.4-P18	4	53	6360	22963	55.204	1296	275	0.053	0.041	0.257	0.397	139.05
D30-S1.4-P19	4	120	14400	49080	148.004	1256	275	0.046	0.035	0.257	0.394	375.64
D30-S1.4-P20	4	163	19560	80153	245.939	1266	326	0.051	0.026	0.269	0.406	605.76
Average	3.9	36.7	4324.5	16148.0	55.540	1224.1	282.3	0.050	0.037	0.128	0.258	186.83

Table 7
Results of *SRP*1 and *SRP*2 on medium-sized instances of set 2.

		3-index formulation (*SRP*1)					Graph-based formulation (*SRP*2)				
Instance (1)	*K* (2)	No. of Binary Var. (3)	No. of Constraints (4)	LB (5)	UB (6)	Time (s) (7)	No. of Variables (8)	No. of Constraints (9)	Optimal Value (10)	Time (s) (11)	Speedup factor (7)/(11)
D30-S1.4-P01	648	194400	280236	232	–	1763.98	4573	966	392	1.104	1597.8
D30-S1.4-P02	634	190200	372458	292	–	2529.54	3040	805	949	0.571	4430.0
D30-S1.4-P03	630	189000	271830	747	–	2179.01	4735	1012	1117	0.925	2355.7
D30-S1.4-P04	643	192900	316013	167	–	1787.92	4886	964	367	0.886	2018.0
D30-S1.4-P05	623	186900	323014	424	–	2235.62	4883	994	517	1.152	1940.6
D30-S1.4-P06	617	185100	318055	202	–	1120.50	5089	1035	202	1.587	706.0
D30-S1.4-P07	643	192900	277433	516	–	1490.28	4735	1012	712	0.977	1525.4
D30-S1.4-P08	630	189000	255450	532	–	TL	4676	983	740	0.774	9302.3
D30-S1.4-P09	603	180900	227028	880	1323	TL	4391	940	1289	1.119	6434.3
D30-S1.4-P10	608	182400	262348	400	–	1143.78	4735	1012	628	1.601	714.4
D30-S1.4-P11	652	195600	266316	841	–	2057.72	4398	920	1356	0.847	2429.4
D30-S1.4-P12	660	198000	212820	38	118666	3026.35	6933	1225	45	2.872	1053.7
D30-S1.4-P13	595	178500	226400	411	141176	TL	4328	873	690	0.664	10843.4
D30-S1.4-P14	623	186900	304916	341	–	1934.20	4883	994	708	1.355	1427.5
D30-S1.4-P15	631	189300	310121	178	148322	TL	4185	872	334	1.174	6132.9
D30-S1.4-P16	617	185100	268078	460	–	2316.47	4440	947	797	0.974	2378.3
D30-S1.4-P17	629	188700	186484	199	99836	TL	6999	1215	266	3.349	2149.9
D30-S1.4-P18	640	192000	276140	793	–	1613.76	4735	1012	823	0.868	1859.2
D30-S1.4-P19	639	191700	261012	348	–	1155.72	4398	920	408	1.234	936.6
D30-S1.4-P20	630	189000	309630	237	99568	TL	4185	872	380	1.471	4894.6
Average	629.8	188925.0	276289.1	411.9	101481.8	3477.74	4761.4	978.7	636.0	1.275	3256.5

"-": no feasible solution found before the time limit (TL = 7200s) or when out-of-memory; LB: Lower bound; UB: Upper bound.

solved to optimality for all instances in only a few seconds. The improvement in computation time of *SRP*2 over *SRP*1 is significant on these medium-sized instances with speedups ranging from 706 to 10843.4. The average speedup factor is 3256.5 allowing instances on average being solved to optimality in slightly more than one minute for *SRP*2 compared to almost one hour for *SRP*1.

Comparing the results of Tables 6 and 7 shows that the size of the graph-based formulation depends on the structure of the work-rule constraints and is independent of the number of staff. The work-rule constraints of Musliu (2006) can be efficiently formulated by the graph-based formulation (*SRP*2) which enables solving instances which are otherwise almost impossible to solve using the traditional 3-index formulation (*SRP*1).

5.2.2. Evaluating SRP*2 performance on large scale instances*

Since Tables 6 and 7 showed that *SRP*2 clearly outperforms *SRP*1, the remaining computational tests focus on analysing the efficiency and effectiveness of *SRP*2 on more challenging randomly generated instances having a planning horizon drawn from {30, 60, 90, 120, 150, 180}, a staffing ratio in {1.2, 1.4, 1.6, 1.8} and a work-rule set in {P01, P02,..., P20}. The results on these instances are summarized in Table 8.

Table 8 reports the time used in constructing the underlying graph without graph simplifications (*construction time*), the number of edges and vertices of the underlying graph before simplifications, the time used in simplifying the underlying graph (*simplification time*), the number of edges and vertices of the underlying graph after simplifications, the time used in solving *SRP*2

Table 8
Average results of *SRP*2 on 480 large-sized instances of set 2.

	Planning horizon (days)					
	30	60	90	120	150	180
No. of instances	80	80	80	80	80	80
Construction time(s)	0.11	0.24	0.40	0.56	0.73	0.93
No. of edges (before)	5664.15	12084.15	18504.15	24924.15	31344.15	37764.15
No. of vertices (before)	1297.10	2812.10	4327.10	5842.10	7357.10	8872.10
Simplification time(s)	0.07	0.46	1.43	3.21	6.08	10.28
No. of edges (after)	4161.35	9007.85	13854.35	18700.85	23547.35	28393.85
No. of vertices (after)	678.65	1476.65	2274.65	3072.65	3870.65	4668.65
MIP time(s)	0.81	3.77	9.61	21.26	29.17	48.25
LP time(s)	0.23	0.42	0.56	0.78	1.09	1.45
No. of variables	4761.35	10207.85	15654.35	21100.85	26547.35	31993.8
No. of constraints	978.65	2076.65	3174.65	4272.65	5370.65	6468.65
Integrality gap(%)	0.0000	0.0037	0.0037	0.0023	0.0034	0.0028
Non-integers in LP(%)	4.17	6.41	7.60	7.75	7.81	7.70

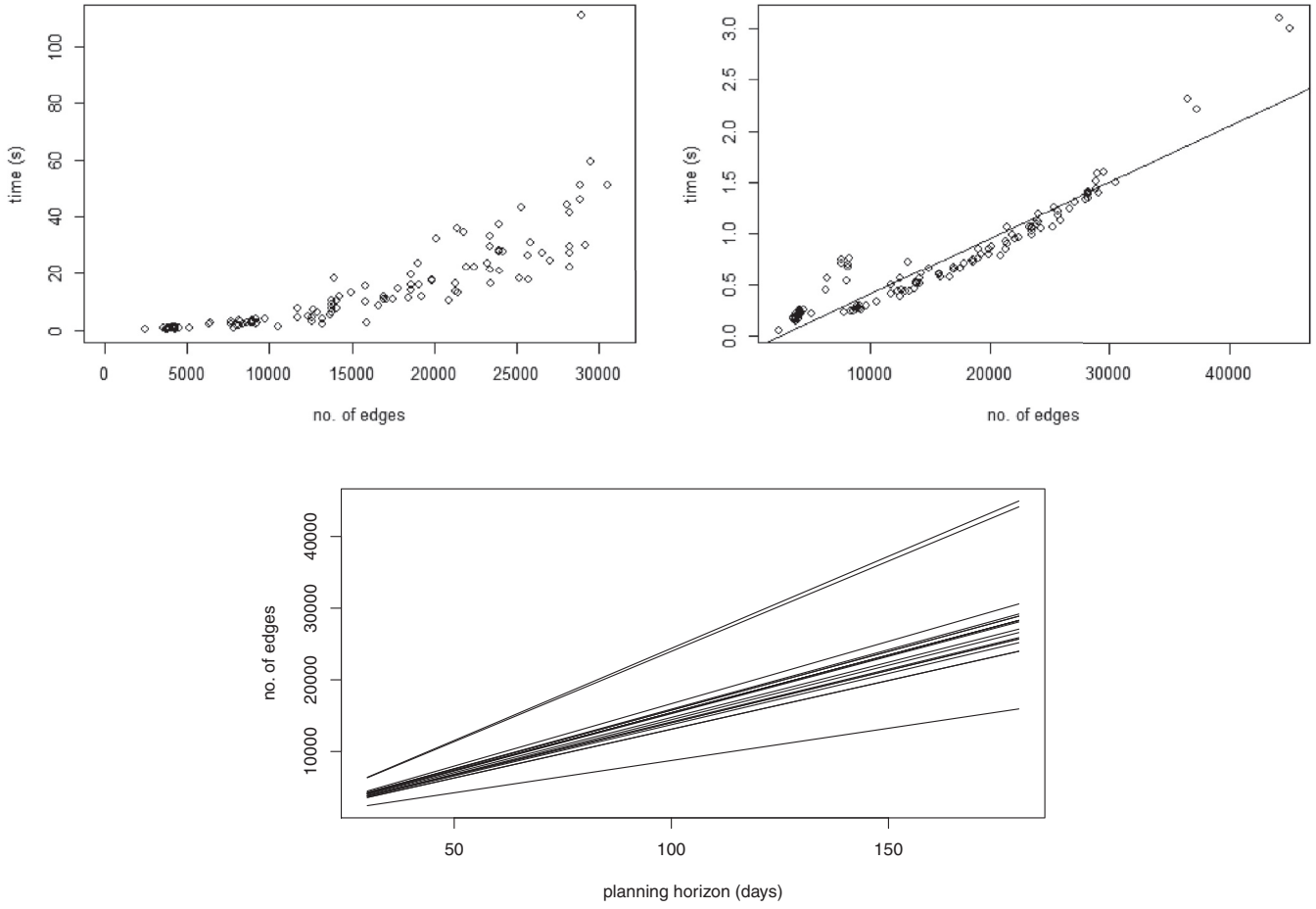

Fig. 11. The time used in solving *SRP*2 (left-hand-side) and its LP-relaxation (right-hand-side). The size of the underlying graph (bottom).

and its LP-relaxation (*MIP time* and *LP time*, respectively), the number of variables and constraints of *SRP*2, the integrality gap and the percentage of non-integers in the solution to the LP-relaxation. Optimal solutions to *SRP*2 were obtained for all the instances.

As shown in Table 8, the underlying graphs of *SRP*2 can be constructed in a few seconds (the worst case is 1.3 minutes) which is negligible compared to the time in solving the corresponding MIP models. The graph size is linearly increasing with the planning horizon as illustrated in both Table 8 and Fig. 11 (bottom). This figure shows the number of edges of the graph as a function of the length of the planning horizons for all instances, with instances of the same set of work rules connected by a straight line. Furthermore, the integrality gap (between the optimal solutions of the LP relaxations and the integer optimal solution of the MIP models) is small (see Table 8) and the time for solving the LP-relaxation of *SRP*2 is linearly increasing with the number of edges of the underlying graph (see Fig. 11 - right-hand-side). This explains why some large rostering problems can be solved by using *SRP*2 and a

Table 9
Average time (s) used in solving *SRP*2 for different staffing ratios and different planing horizons (days) for the 480 large-sized instances of set 2.

staffing	Integer model *SRP*2						LP-relaxation of *SRP*2					
	Planning horizon (days)						Planning horizon (days)					
ratio	30	60	90	120	150	180	30	60	90	120	150	180
1.2	0.77	3.86	8.98	21.06	27.11	48.42	0.36	0.58	0.47	0.73	1.04	1.29
1.4	0.84	4.80	10.06	25.01	31.34	56.52	0.20	0.43	0.57	0.81	1.10	1.52
1.6	0.94	2.65	10.83	21.46	33.20	45.85	0.15	0.33	0.60	0.81	1.13	1.52
1.8	0.69	3.75	8.59	17.53	25.01	42.20	0.18	0.34	0.59	0.79	1.09	1.46

Table 10
Average results of *SRP*2 on 80 instances with random meta-sequences of set 3.

No. of instances	The length of the meta-sequences				
	3 16	4 16	5 16	6 16	7 16
Construction time(s)	0.15	0.60	1.42	2.53	4.09
No. of edges (before)	10392.88	39149.94	90228.75	158926.88	239108.62
No. of vertices (before)	1714.50	5226.00	10920.25	18237.75	26458.56
Simplification time(s)	0.05	0.90	11.07	86.97	468.10
No. of edges (after)	9908.13	36637.94	82335.06	140066.00	202527.69
No. of vertices (after)	1485.19	4339.56	8422.50	13144.81	17979.13
Out-of-memory (No. of instances)	0	0	0	0	2
MIP time(s)	4.58	41.93	252.20	1169.60	1843.18*
LP time(s)	0.35	3.00	12.05	35.52	73.39
No. of variables	10508.13	37237.94	82935.06	140666.00	203127.69
No. of constraints	1785.19	4639.56	8722.50	13444.81	18279.13
Integrality gap(%)	0.03	0.00	0.00	0.00	12.50*
LB	38.19	44.56	24.63	20.69	25.88
UB	38.19	44.56	24.63	20.69	26.81

"*": excluding the out-of-memory instances.

standard commercial solver. Fig. 11 (left-hand-side) also suggests that the time required to solve the MIP formulation of SRP2 (MIP time) increases exponentially with the number of edges.

To examine the impact of different staffing levels, Table 9 reports the average time used in solving *SRP*2 and the corresponding LP-relaxation for different staffing ratios. The staffing ratios are indicated in the rows while the planning horizons are shown in the columns. The results show that *SRP*2 gives a steady performance under the different staffing levels as computational times range for less than a second to under one minute for each of the cases considered.

5.3. Experiment 3

In the third experiment we test *SRP*2 on 80 instances with work rules specified by 5 to 20 randomly generated prohibited meta-sequences. We consider instances with 30 days and a 1.4 staffing ratio. The length of the prohibited meta-sequence ranges from 3 to 7 and every component in the prohibited meta-sequence consists of exactly 3 different shifts randomly selected from a set of 10 shifts. This meta-sequence indicates which sequence of shifts is not allowed and therefore it expresses a work rule, albeit a randomly generated one that cannot be directly related to common work rules such as the fact that a day-off should be assigned after a night shift (see e.g. Table 3). The average results for each prohibited meta-sequence length are summarized in Table 10. Only two instances with meta-sequences of length 7 could not be solved to optimality and are therefore excluded from the calculation of the time used in solving *SRP*2 ("MIP time") and the integrality gap for that problem category. Table A (online supplementary material) presents the optimal solutions and computation time for all the instances for varying number of prohibited meta-sequences.

Both Tables 10 and A (online supplementary material) show that the underlying graph grows as the length of the prohibited meta-sequence increases. This matches our analysis on the graph construction algorithm in Sections 4.3 and 4.4. This effect is illustrated in Fig. 12 (left-hand-side) by plotting the number of edges as a function of the number of prohibited meta-sequences. Each line in the figure connects instances where the length of the meta-sequences are the same. The number of edges increases approximately linearly with the number of prohibited meta-sequences for a given length. The underlying graph is smaller with shorter prohibited meta-sequences (more restrictive work rules) and fewer prohibited meta-sequences (fewer work rules). Fig. 12 (right-hand-side) also shows that the average computation time is more sensitive to the length of the prohibited meta-sequences than to the number of prohibited meta-sequences.

5.4. Experiment 4

In the fourth experiment, we test *SRP*2 on dataset 4 (the NSPLIB instances introduced by Vanhoucke and Maenhout (2007) and Vanhoucke and Maenhout (2009)[3]). The instances have a slightly different problem setting than the first three experiments: staff can have different preferences regarding shifts and days, both resource constraints and prohibited metasequences are needed, and coverage requirements are handled as hard constraints. Therefore, *SRP*2 was adapted as follows:

(i) First, one graph is constructed for each staff member, and the model is used to select an (s, t)-path for each staff member in the corresponding graph, instead of determining a network flow on a single graph for all the staff. This results in a multi-commodity flow formulation with side constraints where staff preferences can be formulated as costs in the objective function.
(ii) Secondly, work rules related to consecutive shift assignments (i.e. the same types of work rules that are tested in the previous experiments) are formulated as prohibited

[3] http://www.projectmanagement.ugent.be/research/personnel_scheduling/nsp.

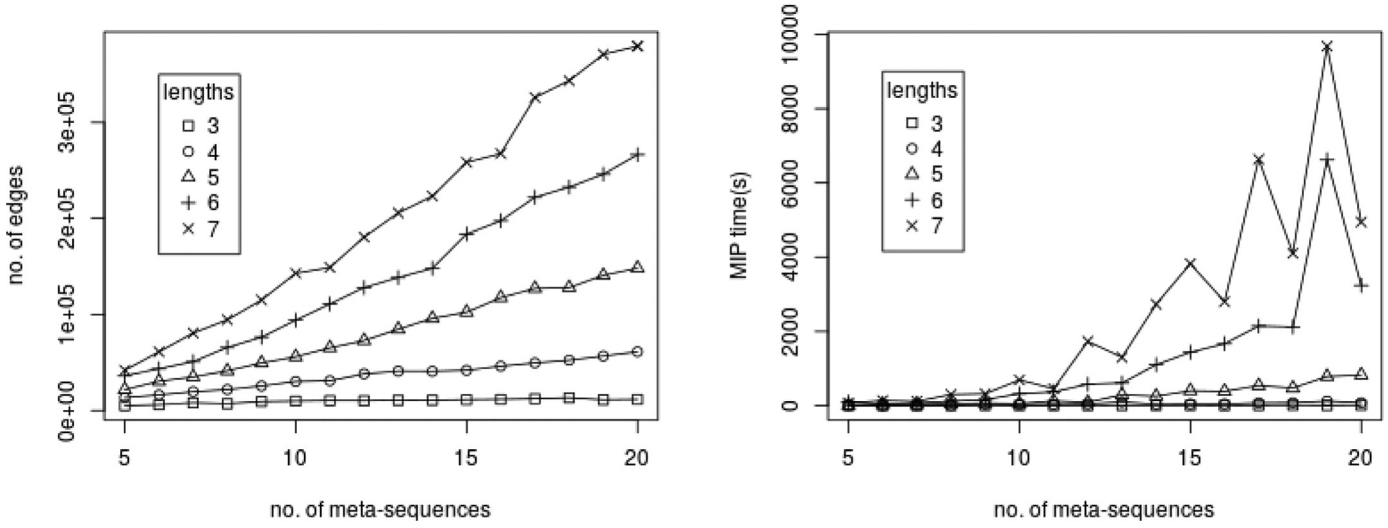

Fig. 12. The size of the underlying graph (left-hand-side) and the average time (right-hand-side) with different number and length of prohibited meta-sequences.

meta-sequences and handled using the graph construction algorithm. The other work rules that cannot be easily formulated as prohibited meta-sequences are handled as resource constraints and are introduced to *SRP*2 as side constraints.

(iii) Lastly, over-staffing penalty costs are set to zero and under-staffing penalty costs are set to a sufficiently large number[4], so that unnecessary under-staffing are avoided.

There are 248640 instances in total which are grouped in two different sets: a so-called diverse set and a realistic set. Each sub-set is characterized by systematically varied levels of the complexity indicators described in Vanhoucke and Maenhout (2007). The types of work rules used in NSPLIB are frequently appeared in the literature, and are extended from the nine constraint types identified by Cheang, Li, Lim, and Rodrigues (2003). The diverse set contains instances with a planning horizon of 7 days and 4 shifts per day, with work rule cases 1 - 8, and the number of staff could be 25, 50, 75 or 100 (denoted as N25, N50, N75 and N100, respectively). The diverse set contains 7290 instances for each work rule case and number of staff, and in total 233280 instances. The realistic set contains instances with a planning horizon of 28 days and 4 shifts per day, with work rule cases 9 - 16, and the number of staff could be 30 or 60 (denoted as N30 and N60, respectively). The realistic set contains 960 instances for each work rule case and number of staff, and in total 15360 instances.

Table 11 summarizes the computational results of experiment 4 for the diverse set. Columns “Avg. Pref.” and “Avg. Pen.” show the average values of the total preference cost and the penalty values (i.e. the total under-staffing penalty cost) of the optimal solutions. Because under-staffing is not allowed in the NSPLIB instances, solutions with under-staffing are therefore considered infeasible. Column “%Feas” reports the percentage of instances where there is no under-staffing in the optimal solutions. Column “Avg. CPU” has the average total CPU time (in seconds) for obtaining an optimal solution including the CPU time for constructing the graphs and solving the MIP models. Column “#Optimal” reports the number of optimal solutions obtained. All instances are solved to optimality within 2 hours CPU time except 7 instances[5]. As for these instances no feasible solution was found, no values for Avg. Pref., Avg. Pen. and %Feas can be reported, but the computation time is taken into account in Table 11.

Table 12 summarizes the computational results of experiment 4 for the realistic set. Although experiments 2 and 3 already indicated that *SRP*2 outperforms *SRP*1 for the large-sized instances of dataset 2 and the instances of dataset 3 (Sections 5.2 and 5.3, respectively), we also want to compare their performance on the realistic set by Vanhoucke and Maenhout (2007). *SRP*1 is unable to solve 34 instances of N30 and 90 instances of N60 whereas *SRP*2 is able to solve all instances except for 1 instance of N30 and 2 instances of N60[6]. Case 15 is more difficult to solve for *SRP*1 and *SRP*2. This might be due to the fact that it has more constrained set of work rules. For example, the number of assignments has to be exactly 20 while in the other cases it is allowed to vary between 16 to 20 (or 16 to 24). Moreover, the number of consecutive same working shifts has to be between 2 and 3 in case 15, while in other cases it can between 1 and 4. Although *SRP*2 was not designed to handle the problem characteristics of NSPLIB, *SRP*2 is able to solve more instances to optimality at a lower average CPU time. The average CPU time of *SRP*2 is 66.90% lower than *SRP*1. It is worth noting though that if *SRP*1 is able to solve an instance, then it is faster than *SRP*2. Based on the experiments of Table 12 the performance of *SRP*2 is more robust than the performance of *SRP*1.

The average results of *SRP*2 and the best-known solutions (BKS) reported in the literature See Table B (online supplementary material) are summarized in the last two rows in Tables 11 and 12. Most of the best-known solutions for these instances are non-optimal as they are obtained by heuristic algorithms. The solutions obtained by *SRP*2 are significantly better than the best-known solutions reported in the literature, with 21.62% and 26.24% improvement in preference cost, 14.17% and 42.77% improvement in feasibility (the percentage of instances where there is no under-staffing in the optimal solutions), and 92.40% and 19.02% reduction in CPU time, for the diverse and realistic sets respectively. Furthermore, *SRP*2 is the first to determine proven optimal solutions for almost all of the instances (except for 10 instances out of 248640 instances). Therefore, the results of this experiment indicate that the proposed method is highly effective and efficient.

[4] 1000000 is used in the computational tests

[5] Instances not solved to optimality within 2 hours CPU time with *SRP*2: N75-Case7-4020, N75-Case7-6446, N75-Case8-5605, N100-Case7-1072, N100-Case7-2698, N100-Case7-6953, and N100-Case8-448

[6] Instances not solved to optimality within 2 hours CPU time with *SRP*2: N30-Case15-113, N60-Case15-174, and N60-Case15-233

Table 11
Computational results for the 248640 benchmark instances of set 4 (Diverse set) using SRP2.

	SRP2 N25					SRP2 N50				
	Avg. Pref.	Avg. Pen.	%Feas	Avg. CPU	#Optimal	Avg. Pref.	Avg. Pen.	%Feas	Avg. CPU	#Optimal
Case 1	245.41	0.00	100.00%	0.01	7290	489.77	0.00	100.00%	0.03	7290
Case 2	231.92	0.00	100.00%	0.01	7290	463.26	0.00	100.00%	0.03	7290
Case 3	256.56	0.00	99.96%	0.02	7290	507.55	0.01	99.92%	0.05	7290
Case 4	244.25	0.00	100.00%	0.01	7290	487.69	0.00	100.00%	0.03	7290
Case 5	247.58	0.00	100.00%	0.20	7290	493.90	0.00	99.96%	0.47	7290
Case 6	233.48	0.00	100.00%	0.05	7290	466.77	0.00	100.00%	0.14	7290
Case 7	263.30	0.13	96.68%	0.45	7290	520.76	0.20	97.60%	1.19	7290
Case 8	241.53	0.03	99.04%	0.20	7290	481.59	0.04	99.59%	0.50	7290
Average	245.50	0.02	99.46%	0.12	7290	488.91	0.03	99.63%	0.30	7290
BKS	257.72	66.72	86.59%	2.16	N.A.	511.13	122.73	87.35%	5.21	N.A.
	SRP2 N75					SRP2 N100				
	Avg. Pref.	Avg. Pen.	%Feas	Avg. CPU	#Optimal	Avg. Pref.	Avg. Pen.	%Feas	Avg. CPU	#Optimal
Case 1	740.11	0.00	100.00%	0.04	7290	1191.19	0.00	100.00%	0.07	7290
Case 2	712.97	0.00	100.00%	0.04	7290	1141.60	0.00	100.00%	0.06	7290
Case 3	767.56	0.01	99.86%	0.09	7290	1248.07	0.00	99.99%	0.13	7290
Case 4	733.65	0.00	100.00%	0.05	7290	1182.14	0.00	100.00%	0.07	7290
Case 5	746.43	0.00	99.93%	0.85	7290	1200.62	0.00	100.00%	1.40	7290
Case 6	715.77	0.00	100.00%	0.24	7290	1146.51	0.00	100.00%	0.36	7290
Case 7	790.46	0.30	97.22%	5.64	7288	1270.66	0.44	97.50%	6.24	7287
Case 8	736.69	0.07	99.30%	1.94	7289	1181.74	0.08	99.36%	4.09	7289
Average	742.96	0.05	99.54%	1.11	7289.63	1195.32	0.06	99.61%	1.55	7289.50
BKS	778.67	191.29	86.81%	11.64	N.A.	1248.00	233.77	88.05%	21.62	N.A.

Table 12
Computational results for the 248640 benchmark instances of set 4 (Realistic set) using SRP1 and SRP2.

	SRP1 N30					SRP2 N30				
	Avg. Pref.	Avg. Pen.	%Feas	Avg. CPU	#Optimal	Avg. Pref.	Avg. Pen.	%Feas	Avg. CPU	#Optimal
Case 9	1424.81	0.00	100.00%	0.28	960	1424.81	0.00	100.00%	8.67	960
Case 10	1366.96	0.00	100.00%	0.25	960	1366.96	0.00	100.00%	5.72	960
Case 11	1492.11	0.00	100.00%	0.37	960	1492.11	0.00	100.00%	17.29	960
Case 12	1411.71	0.00	100.00%	0.26	960	1411.71	0.00	100.00%	7.82	960
Case 13	1442.39	0.03	99.38%	0.95	960	1442.39	0.02	99.48%	11.96	960
Case 14	1384.46	0.02	99.48%	0.32	960	1384.46	0.02	99.48%	4.88	960
Case 15	1530.10	0.99	91.15%	549.98	926	1534.16	1.32	87.60%	38.69	959
Case 16	1442.72	0.78	90.31%	1.99	960	1442.84	0.78	90.31%	7.36	960
Average	1436.91	0.23	97.54%	69.30	955.75	1437.43	0.27	97.11%	12.80	959.88
BKS	1515.50	472.50	67.66%	22.17	N.A.	1515.50	472.50	67.66%	22.17	N.A.
	SRP1 N60					SRP2 N60				
	Avg. Pref.	Avg. Pen.	%Feas	Avg. CPU	#Optimal	Avg. Pref.	Avg. Pen.	%Feas	Avg. CPU	#Optimal
Case 9	2919.66	0.01	99.90%	0.71	960	2919.66	0.01	99.90%	30.86	960
Case 10	2798.91	0.01	99.90%	0.68	960	2798.91	0.01	99.90%	17.06	960
Case 11	3049.43	0.01	99.90%	0.97	960	3049.44	0.01	99.90%	67.14	960
Case 12	2888.45	0.01	99.90%	0.71	960	2888.46	0.01	99.90%	30.13	960
Case 13	2953.86	0.06	99.27%	2.83	960	2953.88	0.06	99.27%	44.79	960
Case 14	2834.26	0.06	99.27%	0.81	960	2834.28	0.06	99.27%	13.67	960
Case 15	3111.40	0.89	95.94%	1,044.82	874	3135.60	2.45	90.10%	216.42	958
Case 16	2950.08	1.31	92.50%	39.81	956	2950.97	1.34	92.29%	22.25	960
Average	2938.26	0.29	98.32%	136.42	948.75	2941.40	0.49	97.57%	55.29	959.75
BKS	3119.59	828.88	68.70%	61.92	N.A.	3119.59	828.88	68.70%	61.92	N.A.

6. Conclusions

Following the call of Smet et al. (2016a) for studying shift rostering problems with generalized work-rule constraints, this paper proposes a novel graph-based network formulation and a specialized graph construction algorithm for the shift rostering problem with generalized work-rule constraints while also handling counter and succession constraints that have been the subject of previous research (Smet et al., 2016a). Coverage constraints are modeled as soft constraints while all work rules are handled as hard constraints. Our focus on staff rostering problems with hard constraints is motivated by the occurrence in practice (Doi et al., 2018) and because problems with additional soft constraints are increasingly solved by hybrid approaches in which the first stage requires solving a staff rostering problem with hard constraints only (Doi et al., 2018; Rahimian et al., 2017).

The current paper proposes a graph-based network formulation in which work rules are formulated in terms of prohibited meta-sequences and resource constraints. This provides the flexibility required for modelling the complicated work rules found in practice. Traditional 3-index integer programming formulations can not be solved efficiently if there is a large number of staff and/or a large number of feasible shift patterns involved. In the proposed graph-based formulation, the set of feasible shift patterns is represented by paths of a graph. By proposing a novel path extension operator, we contribute an efficient algorithm (polynomial to the graph size) for incorporating prohibited meta-sequences into an acyclic direct graph structure. As the size of the graph-based

formulation depends on the structure of the work-rule constraints and on the number of days in the planning horizon but not on the number of staff, the formulation can be solved efficiently for large instances using a standard commercial solver. Furthermore, we have noticed that work-rule constraint structures that appear in real-life work rules can be handled efficiently. Computational testing has shown that the graph-based formulation for large problem instances in datasets 1 and 2 can be solved to optimality in less than one minute whereas the traditional 3-index formulation could not be solved within a 2-hour time limit. For medium-sized problem instances that could be solved by both model formulations the proposed graph-based model formulation and graph construction algorithm resulted in speedup factors of several hundreds to several thousands. Furthermore, the proposed method is the first to determine new best-known solutions and proven optimal solutions of the NSPLIB benchmark dataset for almost all of the instances (except for 7 instances out of 248640 instances) at significantly lower CPU times.

Future research can be directed to using the proposed graph-based approach as a sub-routine within hybrid approaches and decomposition algorithms for solving more complex rostering problems. It is also worthwhile examining how the graph-based formulation can accommodate work rules involving soft constraints and extending the proposed approach to cyclic rostering problems. We hope that our findings on the shift rostering problem will encourage researchers to apply the concept of prohibited meta-sequences to tackle other challenging optimization problems.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.ejor.2019.12.019.

References

Andersen, A. R., Nielsen, B. F., Reinhardt, L. B., & Stidsen, T. R. (2019). Staff optimization for time-dependent acute patient flow. *European Journal of Operational Research, 272*(1), 94–105.

Bard, J. F., Binici, C., & deSilva, A. H. (2003). Staff scheduling at the United States Postal Service. *Computers & Operations Research, 30*(5), 745–771.

Bard, J. F., & Purnomo, H. W. (2005). Preference scheduling for nurses using column generation. *European Journal of Operational Research, 164*(2), 510–534.

Blöchliger, I. (2004). Modeling staff scheduling problems. a tutorial. *European Journal of Operational Research, 158*(3), 533–542.

Brucker, P., Qu, R., & Burke, E. (2011). Personnel scheduling: Models and complexity. *European Journal of Operational Research, 210*(3), 467–473.

Bruecker, P. D., Beliën, J., Van den Bergh, J., & Demeulemeester, E. (2018). A three--stage mixed integer programming approach for optimizing the skill mix and training schedules for aircraft maintenance. *European Journal of Operational Research, 267*(2), 439–452.

Brunner, J. O., Bard, J. F., & Köhler, J. M. (2013). Bounded flexibility in days-on and days-off scheduling. *Naval Research Logistics, 60*(8), 678–701.

Burke, E. K., & Curtois, T. (2014). New approaches to nurse rostering benchmark instances. *European Journal of Operational Research, 237*(1), 71–81.

Cappanera, P., & Gallo, G. (2004). A multicommodity flow approach to the crew rostering problem. *Operations Research, 52*(4), 583–596.

Cheang, B., Li, H., Lim, A., & Rodrigues, B. (2003). Nurse rostering problems — A bibliographic survey. *European Journal of Operational Research, 151*(3), 447–460.

Côté, M.-C., Gendron, B., & Rousseau, L.-M. (2007). Modeling the regular constraint with integer programming. In P. Van Hentenryck, & L. Wolsey (Eds.), *Integration of ai and or techniques in constraint programming for combinatorial optimization problems* (pp. 29–43). Berlin, Heidelberg: Springer Berlin Heidelberg.

Cuevas, R., Ferrer, J.-C., Klapp, M., & Muñoz, J.-C. (2016). A mixed integer programming approach to multi-skilled workforce scheduling. *Journal of Scheduling, 19*(1), 91–106.

Dahmen, S., & Rekik, M. (2015). Solving multi-activity multi-day shift scheduling problems with a hybrid heuristic. *Journal of Scheduling, 18*(2), 207–223.

Doi, T., Nishi, T., & Voss, S. (2018). Two-level decomposition based matheuristic for airline crew rostering problems with fair working time. *European Journal of Operational Research, 267*(2), 428–438.

Dolgui, A., Kovalev, S., Kovalyov, M. Y., Malyutin, S., & Soukhal, A. (2018). Optimal workforce assignment to operations of a paced assembly line. *European Journal of Operational Research, 264*(1), 200–211.

Ernst, A., Jiang, H., Krishnamoorthy, M., & Sier, D. (2004). Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research, 153*(1), 3–27.

Fügener, A., Pahr, A., & Brunner, J. O. (2018). Mid-term nurse rostering considering cross-training effects. *International Journal of Production Economics, 196*, 176–187.

Jarray, F. (2009). A 4-day or a 3-day workweeks scheduling problem with a given workforce size. *Asia-Pacific Journal of Operational Research, 26*(5), 685–696.

Lau, H. C. (1996). On the complexity of manpower shift scheduling. *Computers & Operations Research, 23*(1), 93–102.

Maenhout, B., & Vanhoucke, M. (2016). An exact algorithm for an integrated project staffing problem with a homogeneous workforce. *Journal of Sscheduling, 19*(2), 107–133.

Margot, F. (2010). Symmetry in integer linear programming. In *50 years of integer programming 1958–2008* (pp. 647–686). Springer.

Musliu, N. (2006). Heuristic methods for automatic rotating workforce scheduling. *International Journal of Computational Intelligence Research., 2*(4), 309–326.

Örmeci, E. L., Salman, F. S., & Yücel, E. (2014). Staff rostering in call centers providing employee transportation. *Omega, 43*, 41–53.

Rahimian, E., Akartunali, K., & Levine, J. (2017). A hybrid integer programming and variable neighbourhood search algorithm to solve nurse rostering problems. *European Journal of Operational Research, 258*(2), 411–423.

Santos, H. G., Toffolo, T. A. M., Gomes, R. A. M., & Ribas, S. (2016). Integer programming techniques for the nurse rostering problem. *Annals of Operations Research, 239*(1), 225–251.

Smet, P., Brucker, P., Causmaecker, P. D., & Vanden Berghe, G. (2016a). Polynomially solvable personnel rostering problems. *European Journal of Operational Research, 249*(1), 67–75.

Smet, P., Ernst, A. T., & Berghe, G. V. (2016b). Heuristic decomposition approaches for an integrated task scheduling and personnel rostering problem. *Computers & Operations Research, 76*, 60–72.

Taskiran, G. K., & Zhang, X. (2017). Mathematical models and solution approach for cross-training staff scheduling at call centers. *Computers & Operations Research, 87*, 258–269.

Van den Bergh, J., Beliën, Bruecker, P. D., Demeulemeester, E., & Boeck, L. D. (2013). Personnel scheduling: A literature review. *European Journal of Operational Research, 226*(3), 367–385.

Van Den Eeckhout, M., Maenhout, B., & Vanhoucke, M. (2019). A heuristic procedure to solve the project staffing problem with discrete time/resource trade-offs and personnel scheduling constraints. *Computers & Operations Research, 101*, 144–161.

Vanhoucke, M., & Maenhout, B. (2007). NSPLib — A nurse scheduling problem library: a tool to evaluate (meta-) heuristic procedures. In *Operational research for health policy: making better decisions, proceedings of the 31st annual meeting of the working group on operations research applied to health services* (pp. 151–165).

Vanhoucke, M., & Maenhout, B. (2009). On the characterization and generation of nurse scheduling problem instances. *European Journal of Operational Research, 196*(2), 457–467.

Vermuyten, H., Rosa, J. N., Marques, I., Beliën, J., & Barbosa-Póvoa, A. (2018). Integrated staff scheduling at a medical emergency service: An optimisation approach. *Expert Systems with Applications, 112*, 62–76.

Wong, T., Xu, M., & Chin, K. (2014). A two-stage heuristic approach for nurse scheduling problem: A case study in an emergency department. *Computers & Operations Research, 51*, 99–110.

Zheng, Z., Liu, X., & Gong, X. (2017). A simple randomized variable neighbourhood search for nurse rostering. *Computers & Industrial Engineering, 110*, 165–174.